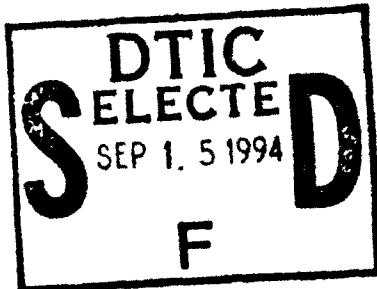


AD-A284 402



CDRL: B015  
31 May 1994

**UNISYS**



## Command Center Library Model Document

Comprehensive Approach to Reusable Defense  
Software (CARDS)

Informal Technical Report

This document has been approved  
for public release and sale; its  
distribution is unlimited.



Comprehensive Approach to Reusable Defense Software

STARS-VC-B015/002/00

31 May 1994

94-29921



DTIC QUALITY INSPECTED 3

94 9 14 063

**INFORMAL TECHNICAL REPORT  
For The  
SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS  
(STARS)**

*Command Center Library Model  
Document Release 4.0  
Comprehensive Approach to Reusable Defense Software  
(CARDS)*

**STARS-VC-B015/002/00  
31 May 1994**

**Data Type: Informal Technical Data  
Contract NO. F19628-93-C-0130  
Line Item 0002AB**

**Prepared for:**

**Electronic Systems Center  
Air Force Material Command, USAF  
Hanscom AFB, MA 01731-2816**

**Prepared by:  
Azimuth, Inc.  
and  
Electronic Warfare Associates, Inc.  
under contract to  
Unisys Corporation  
12010 Sunrise Valley Drive  
Reston, VA 22091**

Accepted	Not Yet Accepted
NTIS	Not Yet
Div	Not Yet
U	Not Yet
Justified	Not Yet
By	
Distribution	
Avail	
Dist	Avail Satisfied
A-1	

**Distribution Statement "A"  
per DoD Directive 5230.24  
Approved for public release, distribution is unlimited**

**INFORMAL TECHNICAL REPORT  
For The  
SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS  
(STARS)**

*Command Center Library Model  
Document Release 4.0  
Comprehensive Approach to Reusable Defense Software  
(CARDS)*

**STARS-VC-B015/002/00  
31 May 1994**

**Data Type: Informal Technical Data**

**Contract NO. F19628-93-C-0130  
Line Item 0002AB**

**Prepared for:**

**Electronic Systems Center  
Air Force Material Command, USAF  
Hanscom AFB, MA 01731-2816**

**Prepared by:**

**Azimuth, Inc.  
and  
Electronic Warfare Associates, Inc.  
under contract to  
Unisys Corporation  
12010 Sunrise Valley Drive  
Reston, VA 22091**

Data Reference: STARS-VC-B015/002/00  
INFORMAL TECHNICAL REPORT  
Command Center Library Model  
Document Release 4.0  
Comprehensive Approach to Reusable Defense Software  
(CARDS)

Distribution Statement "A"  
per DoD Directive 5230.24

Approved for public release, distribution is unlimited

Copyright 1994, Unisys Corporation, Reston Virginia and Azimuth, Inc  
Copyright is assigned to the U.S. Government, upon delivery thereto in accordance with the  
DFARS Special Works Clause

Developed by: Azimuth, Inc and Electronic Warfare Associates, Inc. under contract to Unisys

This document, developed under the Software Technology for Adaptable, Reliable Systems (STARS) program, is approved for release under Distribution "A" of the Scientific and Technical Information Program Classification Schema (DoD Directive 5230.24) unless otherwise indicated by the U.S. Sponsored by the U.S. Advanced Research Projects Agency (ARPA) under contract F19628-93-C-0130 the STARS program is supported by the military services with the U.S. Air Force as the executive contracting agent. The information identified herein is subject to change. For further information, contact the authors at the following mailer address: [delivery@stars.reston.paramax.com](mailto:delivery@stars.reston.paramax.com)

Permission to use, copy, modify, and comment on this document for purposes stated under Distribution "A" and without fee is hereby granted, providing that this notice appears in each whole or partial copy. This document retains Contractor indemnification to the Government regarding copyrights pursuant to the above referenced STARS contract. The Government disclaims all responsibility against liability, including costs and expenses for violation of property rights, or copyrights arising out of the creation or use of this document.

The contents of this document constitutes technical information developed for internal Government use. The Government does not guarantee the accuracy of the contents and does not sponsor the release to third parties whether engaged in performance of a Government contract or subcontract or otherwise. The Government further disallows any liability for damages incurred as the result of the dissemination of this information.

In addition, the Government (prime contractor or its subcontractor) disclaim all warranties with regard to this document, including all implied warranties of merchantability and fitness, and in no event shall the Government (prime contractor or its subcontractor) be liable for any special,

indirect, or consequential damages or any damages whatsoever resulting from the loss of use, data, or profits, whether in action of the contract, negligence, or other tortious action, arising in connection with the use or performance of this document.

Data Reference: STARS-VC-B015/002/00  
INFORMAL TECHNICAL REPORT  
Command Center Library Model  
Document Release 4.0  
Comprehensive Approach to Reusable Defense Software  
(CARDS)

Principal Author(s):

---

*Aleisa Petracca*

*Date*

---

*Tom Bock*

*Date*

Approvals:

---

*System Architect: Kurt Wallnau*

*Date*

---

*Program Manager: Lorraine Martin*

*Date*

*(Signatures on File)*

Data Reference: STARS-VC-B015/002/00  
INFORMAL TECHNICAL REPORT  
Command Center Library Model  
Document Release 4.0  
Comprehensive Approach to Reusable Defense Software  
(CARDS)

## ABSTRACT

This Command Center Library Model Document (CCLMD) was developed under the Comprehensive Approach for Reusable Defense Software (CARDS) Program to help facilitate advances in software reuse methods. It represents the current state of the CARDS Command Center Library Model. It is a "living" document, and will be updated with every Library release. This document describes how software engineering relates and feeds back to Domain Engineering, how Domain Engineering compares and contrasts to Library Modeling, and examines modeling concepts and specialization/aggregation hierarchies.

This document is specific to release 4.0 of the Command Center Library Model in its description of requirements, architectures, qualified components, system composition, and future direction. The intended audience is anyone desiring an understanding of the CARDS Command Center Library Model and wanting a view of the current Library release. A knowledge of software engineering concepts is assumed.

# REPORT DOCUMENTATION PAGE

Form Approved  
OMB No. 0704-0185

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0185), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE 31 May 1994	3. REPORT TYPE AND DATES COVERED Final	
4. TITLE AND SUBTITLE  Command Center Library Model Document (CCLMD) Release 4.0		5. FUNDING NUMBERS  F19628-93-C-0130	
6. AUTHOR(S)  Alesia Petracca			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)  Unisys Corporation 12010 Sunrise Valley Drive Reston, VA 22091		8. PERFORMING ORGANIZATION REPORT NUMBER  STARS-VC-B015/002/00	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)  Department of the Air Force ESC/ENS Hanscom AFB, MA 01731-2816		10. SPONSORING/MONITORING AGENCY REPORT NUMBER  B015	
11. SUPPLEMENTARY NOTES			
12a. DISTRIBUTION AVAILABILITY STATEMENT  DISTRIBUTION "A"		12b. DISTRIBUTION CODE	
13. ABSTRACT ( Maximum 200 words)  This Command Center Library Model Document (CCLMD) was developed under the Central Archive for Reusable Defense Software (CARDS) Program to help facilitate advances in software reuse methods. It represents the current state of the CARDS Command Center Library Model. It is a "living" document, and will be updated with every library release. This document describes how software engineering relates and feeds back to Domain Engineering, how Domain Engineering compares and contrasts to Library Modeling, and examines modeling concepts and specialization/aggregation hierarchies.  This document is specific to release 4.0 of the Command Center Library Model in its description of requirements, architectures, qualified components, system composition, and future direction. The intended audience is anyone desiring an understanding of the CARDS Command Center Library Model and wanting a view of the current library release. A knowledge of software engineering concepts is assumed.			
14. SUBJECT TERMS  Modeling, PRISM, RLP, PRISM, Model-Based reuse library		15. NUMBER OF PAGES 74	16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT  UNCLASSIFIED		18. SECURITY CLASSIFICATION OF THIS PAGE  UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT  UNCLASSIFIED
		20. LIMITATION OF ABSTRACT  SAR	

## Table of Contents

<b>1</b>	<b>Introduction.....</b>	<b>1</b>
1.1	Sources of Input.....	1
1.2	Release Notes 4.0.....	2
<b>2</b>	<b>Domain Engineering and Library Modeling.....</b>	<b>3</b>
2.1	Scoping CARDS Repositories.....	4
2.2	CARDS Methodology.....	6
2.3	AdaKNET.....	7
2.3.1	Specialization.....	8
2.3.2	Individuation.....	8
2.3.3	Aggregation.....	9
2.3.4	The Model.....	10
2.4	Inferencing.....	10
2.5	Actions.....	11
<b>3</b>	<b>Command Center Library Model.....</b>	<b>13</b>
3.1	Scope of the CARDS Command Center Library Release 4.0.....	13
3.2	Background.....	13
3.3	CARDS Command Center Model History .....	13
3.4	Requirements.....	14
3.5	Architectures.....	17
3.5.1	The PRISM Architecture.....	19
3.6	Component Class Models.....	20
3.6.1	Features for that particular component class.....	21
3.6.2	Architectural constraints .....	22
3.6.3	Implementation constraints.....	23
3.7	Qualified Components .....	24
3.7.1	DBMS.....	25
3.7.1.1	Ingres.....	26
3.7.1.2	Oracle.....	27
3.7.2	Briefing System .....	27
3.7.2.1	Lotus_123.....	27
3.7.3	Database User Interface.....	28

<b>3.7.4 Database Front-End.....</b>	<b>28</b>
<b>3.7.5 Multi-Database Database Frontend.....</b>	<b>29</b>
<b>3.7.5.1 SmartStar .....</b>	<b>29</b>
<b>3.7.6 Mapping System.....</b>	<b>29</b>
<b>3.7.6.1 OILSTOCK.....</b>	<b>30</b>
<b>3.7.7 Geographic Information System.....</b>	<b>30</b>
<b>3.7.7.1 GRASS.....</b>	<b>31</b>
<b>3.7.8 Message Translator/Validator Generator.....</b>	<b>31</b>
<b>3.7.8.1 GBMV.....</b>	<b>32</b>
<b>3.7.9 Network Manager.....</b>	<b>32</b>
<b>3.7.9.1 XNetManager.....</b>	<b>33</b>
<b>3.7.10 Office Automation Software.....</b>	<b>33</b>
<b>3.7.11 Electronic Mail Software.....</b>	<b>34</b>
<b>3.7.12 Message Transfer Agent.....</b>	<b>35</b>
<b>3.7.12.1 PP.....</b>	<b>35</b>
<b>3.7.13 User Agent.....</b>	<b>36</b>
<b>3.7.13.1 zmail .....</b>	<b>36</b>
<b>3.7.13.2 XMH.....</b>	<b>36</b>
<b>3.7.14 Spreadsheet.....</b>	<b>37</b>
<b>3.7.14.1 Xspread.....</b>	<b>37</b>
<b>3.7.14.2 lotus123 (spreadsheet).....</b>	<b>38</b>
<b>3.7.14.3 wingz_ss.....</b>	<b>38</b>
<b>3.7.15 Word Processor.....</b>	<b>38</b>
<b>3.7.15.1 Arbortext.....</b>	<b>38</b>
<b>3.7.16 Desktop Publisher.....</b>	<b>39</b>
<b>3.7.16.1 FrameMaker.....</b>	<b>39</b>
<b>3.8 Reference Model.....</b>	<b>39</b>
<b>3.9 System Composition.....</b>	<b>41</b>
<b>3.10 Component Qualification Tool .....</b>	<b>49</b>
<b>3.11 System Demonstrations .....</b>	<b>50</b>
<b>3.11.1 ascii_PRISM_msg_gen .....</b>	<b>50</b>
<b>3.11.2 bb_PRISM_msg_gen.....</b>	<b>50</b>
<b>3.12 Interoperability.....</b>	<b>50</b>
<b>3.12.1 DSRS Interoperability Components.....</b>	<b>51</b>
<b>3.12.1.1 Screen_And_Data_Manager_Package.....</b>	<b>51</b>
<b>3.12.1.2 Generic_Report_Handler.....</b>	<b>51</b>
<b>3.12.1.3 Safe_IO .....</b>	<b>51</b>
<b>3.12.1.4 String_Utils_Package.....</b>	<b>51</b>
<b>3.12.2 ASSET Interoperability Components.....</b>	<b>52</b>
<b>3.12.2.1 Ada_SQL_bindings.....</b>	<b>52</b>
<b>3.12.2.2 Optimization_and_Planning_Tools.....</b>	<b>52</b>

3.12.2.3 Reusable_Image_Processing_Package.....	52
3.12.3 Interoperability Metrics.....	52
3.13 Future Directions/Enhancements.....	53
3.13.1 Structural Changes.....	53
3.13.2 Action Changes.....	53
3.13.3 Anticipated Qualified Components.....	54
3.13.3.1 UNAS_SALE (Universal Network Architecture Service/Software Architect's Lifecycle Environment) .....	54
3.13.4 Anticipated System Demonstrations.....	54
3.13.4.1 WingZ.....	55
3.13.4.2 UNAS_SALE.....	55

## Appendices

Appendix A References.....	A - 1
----------------------------	-------

Appendix B Glossary of Terms.....	B - 1
-----------------------------------	-------

Appendix C LIBRARY ACTIONS.....	C - 1
---------------------------------	-------

## List of Figures

Fig. 2-1 Software and Domain Engineering Process.....	3
Fig. 2-2 Implementation Based Reuse .....	4
Fig. 2-3 Architecture Based Reuse.....	5
Fig. 2-4 Domain Based Reuse.....	6
Fig. 2-5 Specialization Hierarchy .....	8
Fig. 2-6 Aggregation Hierarchy.....	10
Fig. 3-1 requirement concept.....	15
Fig. 3-2 disa_ccdh_item function.....	15
Fig. 3-3 forces_employment_activity .....	16
Fig. 3-4 Aggregational view of plan_execution_7_c.....	17
Fig. 3-5 Relationship between Component Architecture and Services Subsystems.....	19
Fig. 3-6 PRISM Generic Command Center Acrhitecture .....	20
Fig. 3-7 Spreadsheet Feature Tree.....	22
Fig. 3-8 Application Platform Specification Tree.....	23
Fig. 3-9 Hardware Tree.....	23
Fig. 3-10 Application Platform Software Tree .....	23
Fig. 3-11 component_class and its children.....	24
Fig. 3-12 DBMS and its children.....	26
Fig. 3-13 Briefing_system and its child.....	27
Fig. 3-14 Database_User_Interface and its descendants .....	28
Fig. 3-15 mapping_system and its descendants.....	29
Fig. 3-16 message_translater_validator_generator and its child.....	32
Fig. 3-17 network_manager and its child.....	32
Fig. 3-18 office_automation_software and its descendants .....	33
Fig. 3-19 PRISM Technical Reference Model.....	40
Fig. 3-20 application_platform_entity and its descendants.....	41

## 1 Introduction

This Command Center Library (CCL) Model Document describes a version of the Comprehensive Approach for Reusable Defense Software (CARDS) CCL Model. This release replaces earlier versions of this document. For descriptive purposes, this Model is referred to as CCL release 4.0, an encoding of the Generic Command Center Architecture (GCCA) of the July 1992 Portable, Reusable, Integrated Software Modules (PRISM) prototype, into the RLF (Reuse Library Framework) [CARDS93a]. RLF is a part of the CARDS library infrastructure which can be thought of as both a tool and a formalism.

CARDS library models are being iteratively developed. The original CARDS model is being enhanced to reflect changes in the GCCA and to provide additional detail. The command center library model will go through several significant revisions. CCL releases will include updates to this document.

The purpose of this document is to:

- Describe this CCL Model release.
- Describe how this release library model is related to PRISM.
- Provide the reader with an understanding of what CARDS library models represent and how they are related to domain analysis.

Section 2 provides a top-level description of domain engineering and library models which may be skipped if you are familiar with the topics. Section 2 also provides a high-level description of the semantics of the RLF meta-model, i.e., the knowledge-representation formalism used to encode library models. This description is useful for understanding parts of Section 3.

Section 3 describes this release, provides a high-level description of the aspects of the PRISM architecture encoded in the release library, and describes in detail how this release library model is related to the PRISM architecture. Section 3 also describes current plans for evolving this release.

Appendix A is a bibliography of sources used to compile this document.

Appendix B is a glossary of terms used within this document.

Appendix C is a list of actions contained within this release.

Terms having a specific technical meaning are *italicized* the first time they are used and appear in the glossary. Emphasized words appear in **bold**.

### 1.1 Sources of Input

An important requirement for any project attempting to perform a *domain analysis*, or to represent a *generic architecture* for a class of software systems, is that several different system

implementations be examined. For CARDS, so far, the primary input, and the only implemented system, has been the PRISM generic command center. Information from PRISM has included a generic architecture, access to PRISM source code which the CARDS team has examined to uncover low level architectural details, and interaction with PRISM on issues of mutual interest.

CARDS has also used the Defense Information Systems Agency (DISA) Command Center Design Handbook (CCDH) [DISA91] for very high level information, primarily in the area of functional requirements, and also the Department of Defense (DoD) Technical Reference Model [DOD92].

## **1.2 Release Notes 4.0**

The following are additions and changes to the CARDS Command Center Library for Release 4.0. The additions follow the last release of the CARDS Command Center Library 3.3, on 25 February 1994.

Section 3.6, previously Qualified Components, now discusses the Component Classes. The following sections are now organized:

3.7 Qualified Components

3.8 Reference Models

3.9 System Composition

3.10 Component Qualification Tool

3.11 System Demonstrations

3.12 Interoperability

3.13 Future Directions/Enhancements

Section 6.0, Library Actions, has been modified. The definitions are provided for each of the actions, but the list of nodes at which these actions occur has been removed.

## 2 Domain Engineering and Library Modeling

Although reuse may be approached in a variety of informal ways, CARDS position is that a formal, systematic integration of reuse into the conventional software development process yields substantially greater rewards. The basis for this increased formality is the emerging disciplines of domain analysis and domain engineering. Figure 2-1 illustrates the corresponding relationships between aspects of domain engineering and software engineering. The current state of research and practice of domain analysis and domain engineering are well represented in an IEEE tutorial, Domain Analysis and Software Systems Modeling [PRIE92]. The key points are that:

- Domain Engineering targets a well-defined application area (or *domain*) and results in the creation of various products characterizing this domain (e.g., the *domain model*).
- A key step in domain engineering is domain analysis, which is roughly analogous to software engineering requirements analysis, except that domain analysis describes the requirements of a family of systems (i.e., the requirements of the domain), while software engineering requirements analysis focuses on the needs of a particular system in question.

A domain model may encompass not only the results of the domain analysis (as just defined), but may also include other aspects of the domain as well, including a generic architecture for systems within the domain, and reusable components that satisfy the conditions of the generic architecture.

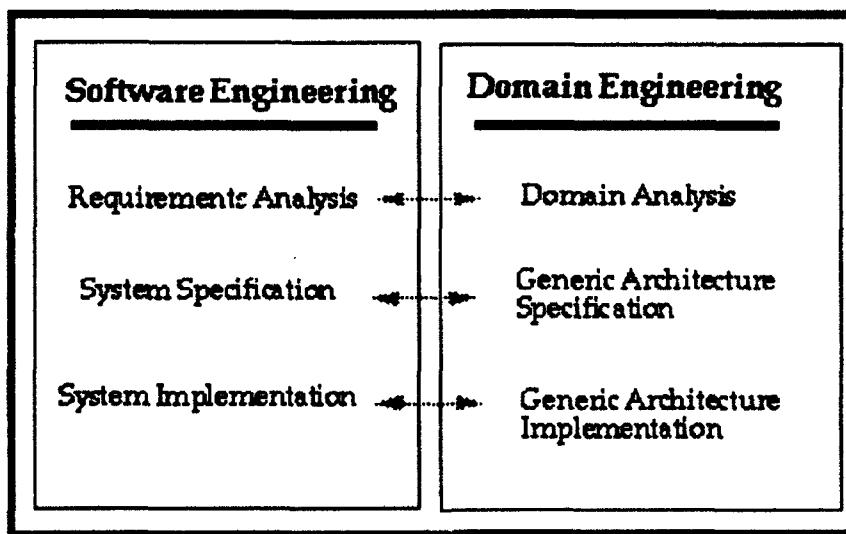


Figure 2-1 Software and Domain Engineering Process

This itemization is not meant as a precise and fixed definition of domain engineering (since this is a still-emerging discipline), but rather as a basis for understanding the relationship of CARDS libraries to both domain engineering and software engineering processes.

Note that in the above summary the terms domain analysis, domain model and generic architecture are used, but the terms *library model* and *library modeling* are conspicuously absent. The reason for distinguishing library models and modeling from domain models and modeling is twofold.

- Since CARDS is meant to provide a generic capability for constructing domain-specific reuse libraries [CARDS94d], it is important that CARDS does not preemptively select one set of domain engineering techniques at the expense of others. Since different application domains (and different DoD programs) may be best served by different domain analysis and domain modeling techniques, any such CARDS selection would be premature and counterproductive.
- The decision of which domain engineering by-products to capture, and how to represent them, is a design decision based not only upon the nature of the domain and the domain engineering analysis techniques used, but also on the anticipated use of these products during software engineering. That is, the reuse repository acts as an integrating agent between domain engineering and software engineering processes. The form this integrating agent (i.e., the repository) needs to take is dependent upon both endpoints of the integration relation - domain engineering and software engineering. This point is elaborated in the following section.

## 2.1 Scoping CARDS Repositories

It is possible, as in Figure 2-2, to scope the repository to capture only the reusable components produced as a result of the implementation phase of domain engineering processes. In Figure 2-2, this scoping is denoted as a parts library. The utility of parts libraries without including some form of architectural context may seem limited, but can be justified in cases where the domain architecture is unstable (i.e., still undergoing technological evolution or standardization), or where a relatively small number of components in the library would not justify the investment in maintaining an up-to-date architectural description. Note in Figure 2-2 that reusable parts can be "used" during system implementation, but that an "understanding" of what parts are available in a repository can aid in system specification.

In Figure 2-3, the scope of the repository has been extended to encompass the reusable components as well as an architectural model capturing the relationships among the components, and describes the purpose of the components within an overall system. One advantage of such a library, denoted as generic architecture library in Figure 2-3, is that the additional context information provided for reusable components can support the automatic composition of systems or parts of systems. For example, if a component implementing part of a database subsystem were selected for use in an application, the generic architecture would provide the basis for the automatic selection and retrieval of any components that were implied by the selection of the original component (e.g., SQL interface code peculiar to a particular relational database vendor).

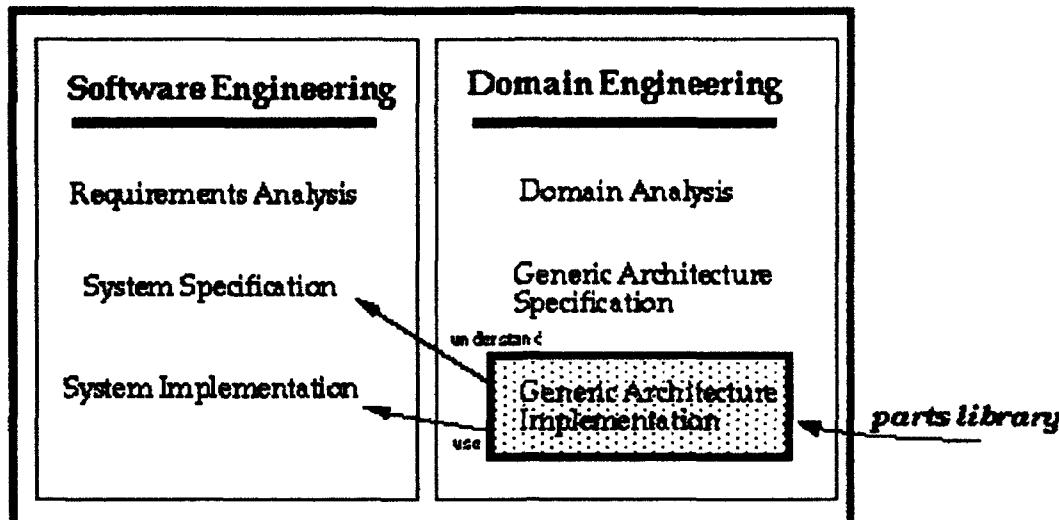


Figure 2-2 Implementation Based Reuse

In addition, a generic architecture library can make the browsing and searching of large libraries simpler and more meaningful to engineers than, for example, faceted classification schemes requiring strict usage of library-specific keywords. Note that in Figure 2-3 the addition of generic architecture information in the library model extends the "use" relation to include both system specification and implementation, while requirements analysis can be aided by "understanding" the architecture supporting applications within a domain.

Figure 2-4, extends the scope of the repository to encompass all of the by-products of domain engineering. This additional scoping broadens the focus of the reuse library to incorporate domain variance as well as domain commonality. While generic architectures focus on what is the same across applications in a domain, a domain model also captures differences across applications in a domain. This broadened focus can support the capture of design rationale for alternative designs of the underlying generic architecture; this, in turn, would be instrumental for evolving the domain architecture in response to new technology and new demands on the previously developed applications.

In Figure 2-4 the advantages of a domain model library are shown both by illustrating the "use" relationships present throughout software engineering processes, and by illustrating the potential feedback loop from software engineering to the domain model. This is done by capturing the variations of each successive system developed from the reuse library and why these systems varied.

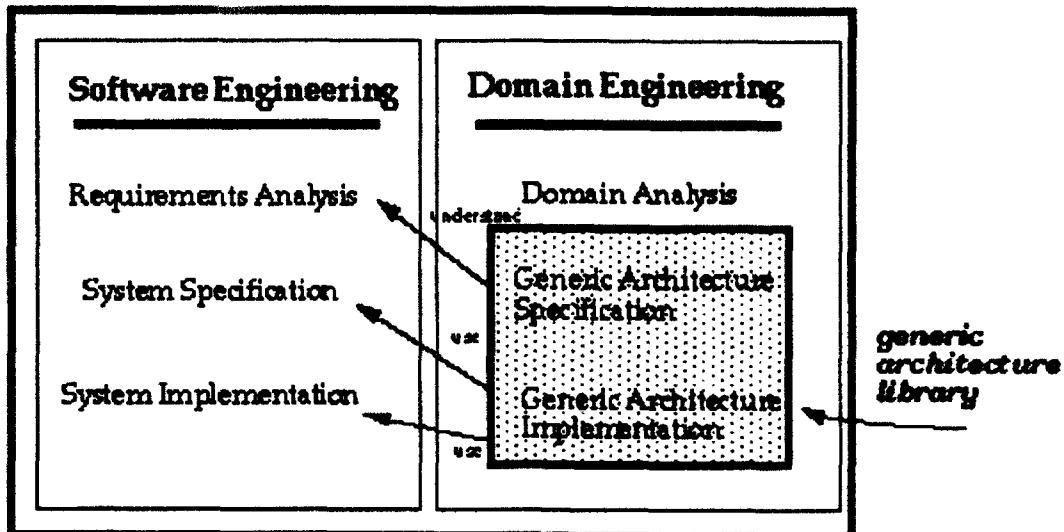


Figure 2-3 Architecture Based Reuse

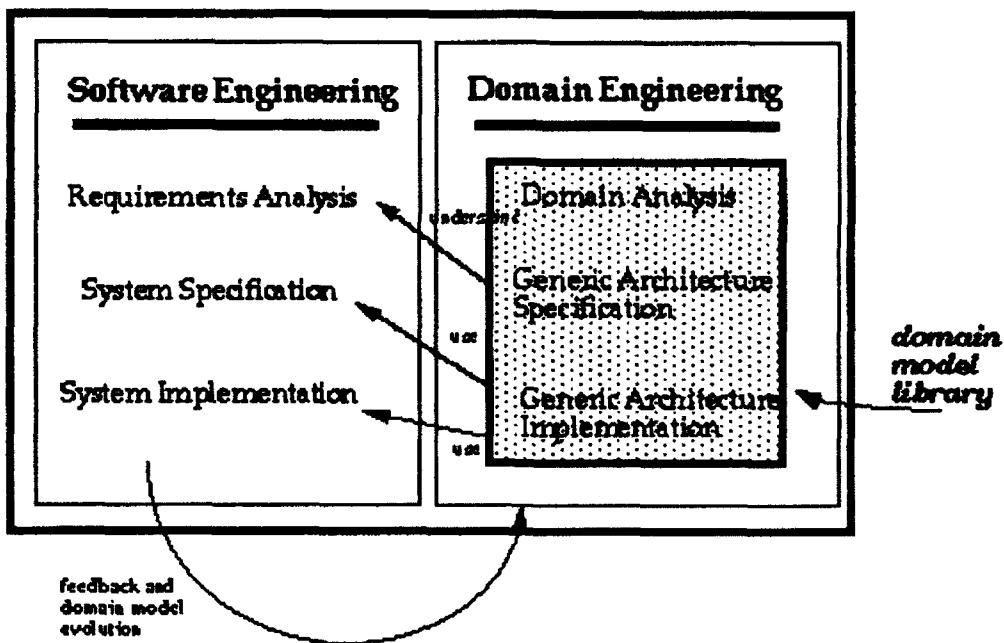


Figure 2-4 Domain Based Reuse

## 2.2 CARDS Methodology

CARDS views a library as a library model and a set of applications, and the construction of a library model as a design activity balancing various requirements. What goes into and what comes out of a library is dependent upon the library modeling formalism used, the kind of domain analysis conducted, and the kind of library applications needing to be constructed to

support the anticipated system engineering processes. The library model includes information in addition to that derived from, or pertinent to domain analysis [CARDS94d].

There are two approaches to implementing a reuse library: *component-based* and *model-based*.

- *Component-based* libraries are organized around a collection of reusable components. The underlying operational concept is that of search and retrieval of individual components. Components found in such libraries are classified in broad, generalized categories.
- *Model-based* libraries use domain models as a foundation for library organization and a framework for supporting applications exploiting these models to automate various library services. Model-based libraries encompass information such as domain knowledge, generic architecture specifications, requirements, and implementation restrictions, as well as software artifacts [CARDS94d].

One of the visible efforts of library analysis is that of organizing the storage and retrieval of the reusable components. CARDS approach is fundamentally model-based (the motivation has more to do with the retrieval of components, and with the capture of reusable information that is lost in component-based libraries, than with storage). The idea is that one of the major obstacles to reuse is the difficulty potential reusers have in locating reusable products. Models are a very powerful mechanism for organizing components and facilitating user access to them.

A CARDS model is a representation of a specific type of application area (often referred to as a domain) which is simply a limited subject area. Because it mirrors the organization of the part of the real world using the components being stored, it provides a means of organizing and accessing a store mechanism. Components are "attached" to the model in the spots representing their use in real world applications. The fact that it mirrors the application area makes it a natural way for the user to locate the components they are interested in. This user friendliness is one of the main motivations for organizing the repository storage and retrieval around the model.

The other motivation is that the model is the starting point for potentially many powerful tools to aid the user in reuse related activities. Because the model is "computationally accessible" and is a detailed picture of what is involved in the domain, it is an excellent starting point for building tools that can "reason" about the various elements involved in the domain, including the specific components being stored in the library. Inference engines are used to analyze the information encoded in the model and apply various types of understanding to produce results such as software configuration.

The initial scope of the CARDS CCL includes both the high level PRISM architecture and descriptions of the components used to implement this model in demonstration prototypes. Thus, this library release can be considered a generic architecture library as depicted in Figure 2-3. As the command center model matures, the CARDS library evolves from a generic architecture type to a domain model type, permitting more powerful tools and information for the user. This domain model library will incorporate more information pertaining to domain analysis and will receive user feedback on the various instantiations of the systems within the library.

Though a domain analysis may be conducted to a large extent without regard to the final form of its products, the knowledge acquired can be more fully realized once it is harnessed by a library model in some formalism. RLF is the mechanism used to implement the CARDS CCL since it integrates a knowledge representation scheme, rule-based inferencing and a graphical browser. The description of the nature of the model requires an overview of the encoding mechanisms.

## 2.3 AdaKNET

RLF has a knowledge representation scheme, called AdaKNET, which facilitates the classification of library components. AdaKNET can be thought of as a graph, in which the nodes represent general categories or specific objects, and the edges represent relationships between the nodes. Nodes, representing general classes of knowledge, are called *concepts* or *categories*. There are two basic types of relationships, *specialization* ("is-a") and *aggregation* ("has-a"), described below. Two kinds of hierarchies are built with the concepts and relationships: specialization and aggregation hierarchies.

### 2.3.1 Specialization

There is exactly one specialization hierarchy in any given "model". The specialization hierarchy is a way of defining the concepts. It can be thought of as a glossary, and can be compared to the need to declare variables in a traditional computer programming language. The top level concept is usually given a very generic name, such as *thing* or *entity*, and all other concepts are defined to be specializations of it. This hierarchy is made by asserting that the "is-a" relationship holds between two concepts. This relationship is unidirectional.

The assertion is that *less\_general\_concept* specializes *more\_general\_concept*. Or, *less\_general\_concept* is-a *more\_general\_concept*.

As depicted in Figure 2-5, *animal* specializes *thing*, *mammal* specializes *animal*, and *cow* specializes *mammal*. The modeler can build a set of concepts representing all the parts of the domain which need to be referenced. In this view, concepts are represented as a single oval and the specialization links are represented as double lined arrows. A concept represents abstract categories of concise things and may also be called a generic concept, a category or a class.

The specialization hierarchy provides the vocabulary for an AdaKNET model. Because every concept in the model is defined in this hierarchy as being a specialization of some other concept, we have a way of understanding the context of any concept we encounter in the model.

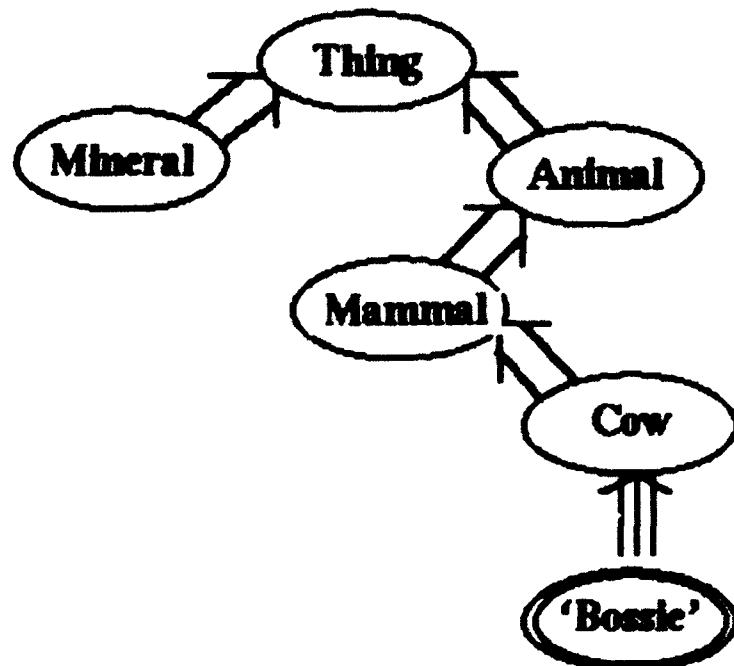


Figure 2-5 Specialization Hierarchy

### 2.3.2 Individuation

Within the specialization hierarchy, the top-level node is very generic and becomes more and more specific at each level until finally the leaf nodes may contain specific examples of their parent concepts. A leaf node representing a particular component or elemental piece of information instantiated from its parent concept is known as an individual or object. It also may be known as an individual concept, or an instance.

Individuation is the relationship between a parent concept and the instantiation (individual) of that parent concept. Individuation lies within the specialization hierarchy and defines an individuation link between a concept and an individual.

Referring to Figure 2-5, take note of the individual concept called 'Bossie'. In this example, 'Bossie' is our specific cow. Yet, 'Bossie' is but one of an infinite number of possibilities for an instantiation of the concept cow. As shown in this Figure, individuals are represented as double ovals and the individuation link is represented as a three lined arrow.

### 2.3.3 Aggregation

Each of the concepts in the specialization hierarchy may have relationships of the "has-a" nature with any other concept in the model, including itself. These relationships may express attributes, characteristics, features (as the term is used in FODA [KANG90]), functional capabilities, requirements or metrics, any relationship that exists between two concepts which is not a

specialization relationship. Since any concept may have aggregation hierarchy under it, and it is not required that all the units of aggregation structure tie together, the model usually contains many separate pieces of aggregation hierarchy. While every concept must appear in the specialization hierarchy, it is not necessary for every concept to be involved in an aggregation relationship.

The aggregation hierarchy is built by beginning with the concept representing the thing being modeled, such as *command\_center*, and listing all of its has-a relationships. These relationships show substructure, characteristics or other sorts of relationships and are often called *roles*. Each of these relationships has a *name*, *type* and *range*. The type is the concept being pointed to. The range is an ordered pair, zero to infinity, or some narrower specification, including a converged range such as 2 to 2. This value represents how many copies of that relationship may/must exist simultaneously.

Each concept automatically *inherits* the aggregation relationships of its ancestors in the specialization hierarchy. AdaKNET supports multiple inheritance, so a concept having more than one parent inherits the aggregation relationships of each. The range, and the possible values of the type, may be narrowed on subsequent levels of the hierarchy (by the concepts that inherited them) to support the logical structure of the aggregation hierarchy. This is called *role restriction*.

Referring to Figure 2-6, notice that the individual 'Bossie' has what is known as a *filler* that satisfies its predecessor's inherited relationship of *makes\_milk* to the type *milk*. Individuals must have such aggregational relationships to other individuals. In this case the individual type is 'Bossie's milk' which is an individuation of the concept *milk*.

### 2.3.4 The Model

These two types of hierarchy are completely intertwined. In most cases what the modeler and the end user think of as "the model" is actually a subtree (really a subgraph, but it is generally thought of as a tree) rooted at the concept representing the thing to be modeled and all the aggregation hierarchy under it.

A full model includes much more than the structural organization expressed in AdaKNET, with the inferencers, discussed below, being the most important other part.

However, in this document the focus is on the model's structural part because the structure is the aspect that is most usefully described. Discussing why the structure was organized as it was communicates something about the modeler's understanding of the command center domain. In contrast, the inferencers are aids for accomplishing tasks. Their actual implementation is irrelevant to the end user.

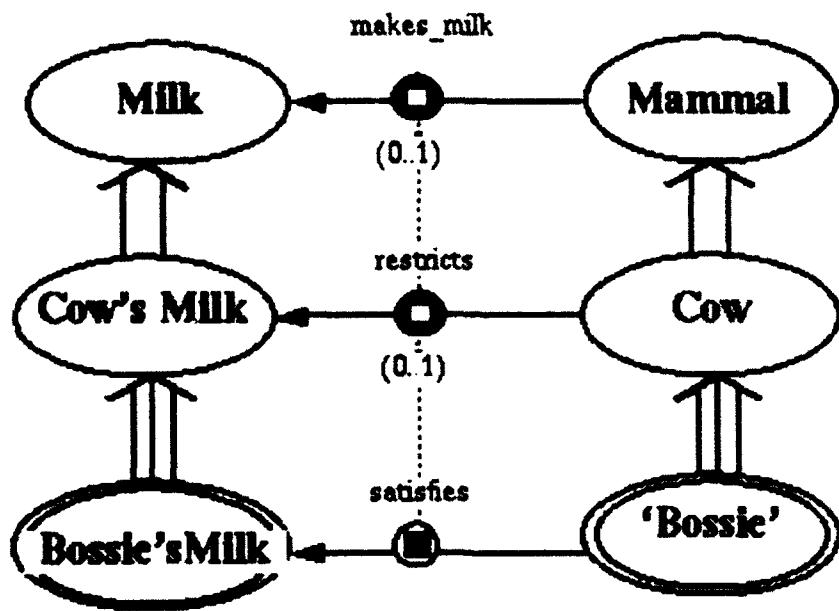


Figure 2-6 Aggregation Hierarchy

## 2.4 Inferencing

There are two inference engines associated with AdaKNET. AdaTAU was written in conjunction with AdaKNET and is a part of the RLF. It is tightly integrated with the Graphical Browser. CLIPS (C Language Integrated Production System)[GIAR92] is used in inferencing tools that work on the AdaKNET structure. CLIPS was developed for the National Aeronautic and Space Administration (NASA) and is available for a very moderate fee, or free for use on Air Force or NASA projects. These two inference engines provide similar basic capabilities, but CLIPS is more computationally powerful and also has the capability of querying the AdaKNET model structure.

Both inference engines permit the modeler to write *inferencers*, units of code expressed as rules about the model. These units, sometimes called *rule-bases*, are associated with specific concepts in the AdaKNET model.

## 2.5 Actions

Any executable program or process, called an "action", may be associated with any concept by the creator of the library. The mechanism of invoking an "action" at a concept is typically used to view textual information associated with the concept, but has general applicability to a wide variety of needs.

Actions can be thought of as strings which are executed in an operating system when the action is invoked. Action "targets" are the strings representing the object upon which the action acts.

For example, a file describing the database management system INGRES may be viewed at the concept *ingres* by including an action which executes the command:

preview *ingres\_desc.txt*

In this example, "*ingres\_desc.txt*" is the action target. The same action, with a different action target, may be made available at the concept *sybase* which executes the command:

preview *sybase\_desc.txt*

Actions are inherited by concepts in much the same manner as roles. All subconcepts are subsumed by a concept declaring an action will have that action available. Actions can also be inherited along several specialization links in the case of multiple inheritance.

Actions, again much like roles, can also be restricted at subconcepts below the concept where they are declared. Action targets can be renamed at subconcepts, regardless of inheritance.

### 3 Command Center Library Model

The CARDS command center model is the basis for a domain-specific software reuse library for the command center domain. The current primary source of information about command centers is the PRISM Program at ESC/ENS, Hanscom AFB. PRISM is designing a generic architecture and developing components for prototyping Air Force command centers. This chapter describes how the PRISM work relates to the current CARDS library model.

#### 3.1 Scope of the CARDS Command Center Library Release 4.0

The initial scope of the CARDS CCL included both the high level PRISM architecture and descriptions of the components used to implement the model in demonstration prototypes.

This scoping remained the primary emphasis by CARDS throughout Phase 2, although some significant extensions of the CARDS CCL to capture useful variant information (i.e., domain model library information) have been produced as a byproduct of the system composition tool and component qualification process development.

CARDS libraries are produced by encoding various results of domain engineering processes into the RLF. The decision about what kinds of domain engineering by-products to encode (i.e., determining the library scope) is a design decision which must consider the nature of the domain, the domain engineering processes, and, especially, the demand-side software engineering processes which will make use of CARDS libraries.

#### 3.2 Background

The Generic Command Center (GCC) Project (the forerunner of PRISM) integrated components for use in command centers. [ESD90] states:

*"The Generic Command Center Phase 2 prototype is an implementation of a portion of the Generic Command Center (GCC) architecture. The purpose of this prototype is to validate the concept of building command centers by integrating large reusable components. The required functionality is that of processing... messages; establishing a database; displaying information in a geographic information system; creating tables of information; and creating briefings that interface with the database. The implementation was consistent with the GCC architecture and used several commercial off-the-shelf (COTS) products as components... The results of the GCC Phase 2 are extremely promising toward the concept feasibility of integrating large software components for application in the Command Center domain."*

PRISM has a more ambitious goal of developing a real generic architecture for command centers. It proposes to provide a user with 80% of the required resources to produce a new command center as well as information on acquiring or producing the remainder. The CARDS library model is incrementally encoding information generated by PRISM.

### 3.3 CARDS Command Center Model History

The initial library model was encoded in RLF while the GCC Phase 2 prototype was being built. The first model, encoded fairly quickly in cooperation with GCC and other ESC/ENS personnel, was available for an Initial Operating Capability (IOC) demonstration in late 1991. The initial model was primitive, but was a useful demonstration of many of the features of the RLF and the CARDS Command Center Library.

Release 2.0 of the Command Center Library included a refinement of the library model made in response to a new release of the operational library software. The model was updated substantially, with major changes to the high level structure of the command center aggregation hierarchy, the addition of the message processing subsystem, and an initial allocation of military requirements to computational subsystems to produce the current model.

Release 3.0 of the model retained the same high level structure as Release 2.0 and differed primarily by addition of lower level concepts to represent new components.

Release 3.1 of the model retained the same high level structure as release 3.0, with the addition of several new components and aggregational relationships for many of the component classes.

Release 3.2 of the model retained all of the features of release 3.1, with the addition of trilateral interoperability, interoperability metrics and system demonstrations. CARDS has implemented interoperability between CARDS, ASSET (Asset Source for Software Engineering Technology), and DSRS (Defense Software Repository System). This addition has prompted the addition of Sections 3.10, Interoperability, and 3.10.1 Interoperability Metrics. The addition of demonstrations opens the door for future demonstrations of various systems in the library and prompted the addition of Sections 3.9 System Demonstrations, and 3.11.4, Anticipated System Demonstrations, to this document.

Release 3.3 of the model reflected modifications to the Command Center Library. These modifications included the addition of new nodes and graphical images, removing and renaming various nodes, and adding and modifying selected relationships associated with specific nodes.

Release 4.0 of the model partitions the CCL model into smaller, more manageable models. The CCL was partitioned into separate models in the following manner: an architectural model which contains the Architecture and Technical Reference Model hierarchies; a requirements model which contains the DISA CCDH and TACE requirements categories and objects; and models for each component class, including sub-component classes for which there are criteria.

CARDS will see an increasing involvement with various organizations through partnerships, both of the command and control and other domains. The goal behind such partnerships is not necessarily support for the command center model, but promotion of reuse adoption. CARDS will also see an increase of work in the area of architectures in two ways:

1. The "fleshing" out of the GCC architectural representation.

2. The investigation of alternative architectural views, including architectures other than the GCC, and tools other than the RLF.

### 3.4 Requirements

The requirements section of the model begins with the requirement node. The requirement node is the ancestor of all military functions and activities required in a command center. The specializations in the requirement subtree divides requirement into disa\_ccdh\_item and tace\_item. disa\_ccdh\_item is specialized by function and activity (Figure 3-1).

There are two ways of looking at the functions and activities in the requirements area of a GCC: (1) as functions and activities that must be contained by a GCC (according to disa\_ccdh) or (2) as a path for finding activities corresponding to functions and the components that must be utilized to perform those activities.

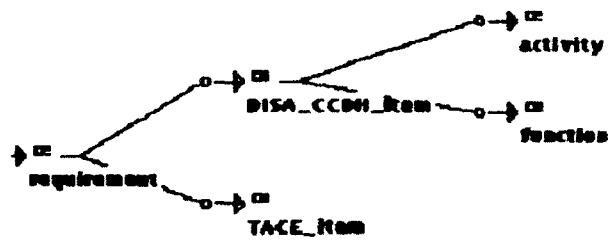


Figure 3-1 requirement concept

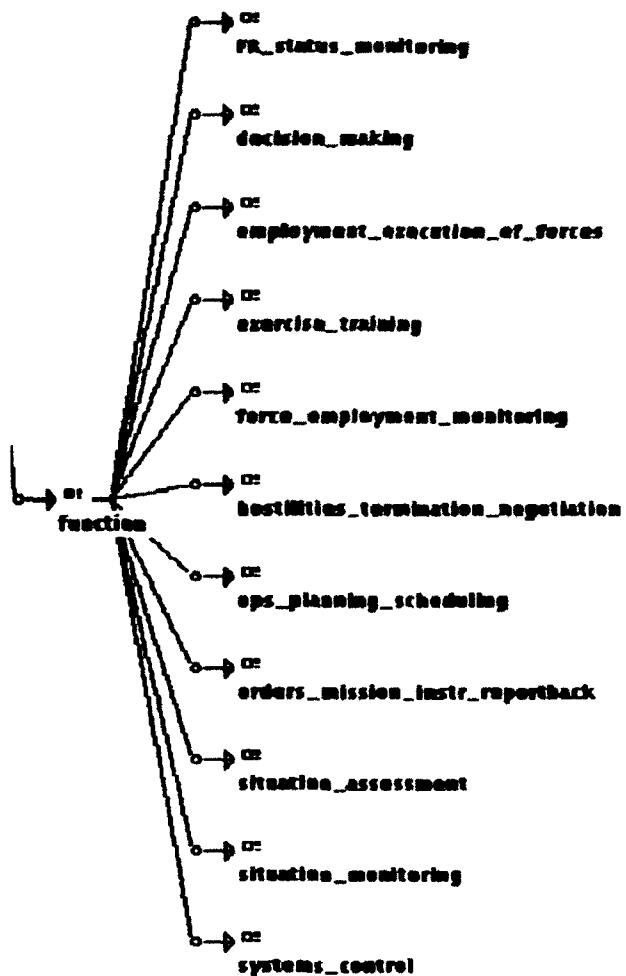


Figure 3-2 disa\_ccdh\_item function

According to the DISA CCDH [DISA91], and augmentation by PRISM, the military functionality of a command center is divided into eleven basic functions. Each of these functions is divided into an arbitrary number of activities, with a total of 76 activities (see Figure 3-2).

The DISA CCDH further breaks down each activity into tasks. These tasks are not represented in the current model as they are at a level of detail not currently necessary for modeling purposes.

Note the 11 DISA CCDH functions (in Figure 3-2) that must exist in a command center. The corresponding activities of the function forces\_employment\_activity are shown in Figure 3-3. These are the activities that must be performed to accomplish forces\_employment function. Figure 3-4 shows the aggregational view of the activity plan\_execution\_7\_c. The links to the components that make up the activity can be seen.

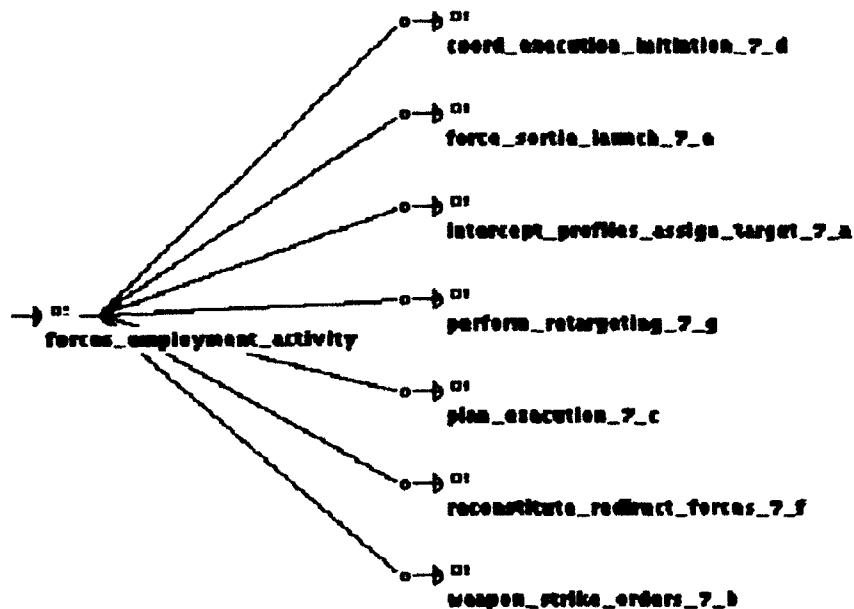


Figure 3-3 forces\_employment\_activity

The TA/CE Command Center Systems Architecture Handbook [DCA90] is another functional decomposition of the command center and is represented by the **tace\_item** in the model. The TA/CE decomposition is also not represented in the model, but the **tace\_item** stub is present for the possibility that its functional decomposition of the command center will be included in the future.

Most of the activities are connected to the software component classes that implement them by aggregation links. These links will permit the system composition tool to determine for the user which components they need, based on the user's specification of what military functionality they wish to have.

### 3.5 Architectures

The CARDS Command Center Architecture is based on the PRISM Command Center Software Architecture. To uncover the CARDS architecture, we must first discuss the PRISM Command Center design.

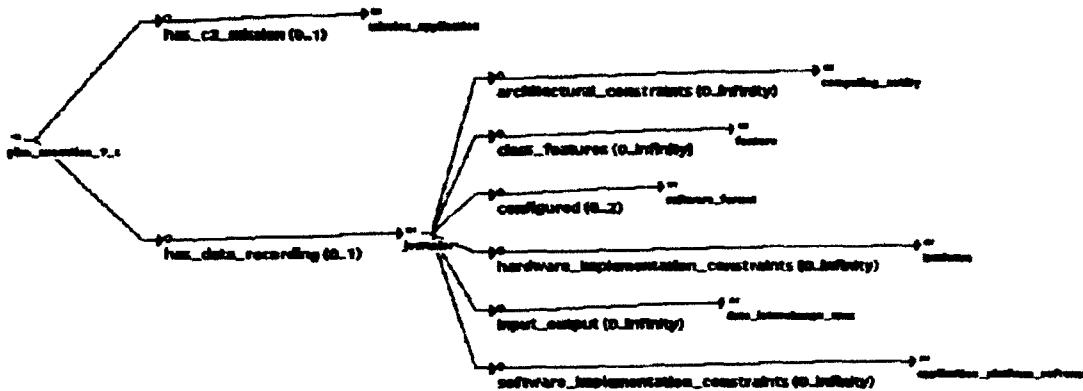


Figure 3-4 Aggregational view of plan\_execution\_7\_c

PRISM has used the DoD Multicommmand Required Operational Capability for Command Centers [DOD91] to decompose a generic command center into its constituent subsystems. This division specifies four subsystems:

- Facility
- Communications
- Information Processing
- Briefing and Display

The Facility Subsystem includes such items as furniture, building components, and the design of the functional spaces, including providing utilities such as heat, light, power, etc.

The Communications Subsystem comprises the hardware, software and protocols required to interconnect equipment within the command center and to terminate external communication links entering the command center. This subsystem includes voice communications but excludes intra-command center communications covered by the Information Processing and Briefing and Display Subsystems.

The Information Processing Subsystem (IPS) is comprised of External Interfaces, Networking, Mission Processing, and Human-Machine Interfaces. The functionality of this subsystem is expected to be physically distributed to the extent permissible within security constraints. Local area networks (LANs) will exist in individual command center work spaces. The number and location of servers and peripheral devices in various command center work spaces will vary depending on security and performance requirements. Gateways and bridges will permit communication among the various LANs and with other automatic data processing (ADP) systems. The IPS is the heart of the command center model.

The Briefing and Display Subsystem (BDS) is comprised of graphics workstations, video switches and controllers, large screen displays and monitors, video teleconferencing, and other audiovisual

support equipment. The BDS is connected to the backbone network. Such connections provide a path for obtaining digitized graphics developed on IPS workstations. Direct connections from selected IPS workstations to the video switch permits presentation of the graphics on large screen displays and monitors which are located in various work spaces throughout the command center.

As in the previous version of the CARDS library, the IPS is still the only subsystem implemented by PRISM and it is the only subsystem discussed in this document.

### 3.5.1 The PRISM Architecture

This section describes the command center software architecture. The components are grouped into four sets of functions:

- External Interfaces
- Mission Processing
- Human-Machine Interface
- Network System

The relationship between the services outlined in the previous section and the functional areas of the architecture is shown in Figure 3-5. As can be seen in this diagram, the functional areas of the architecture are related to the IPS, which itself depends upon both the Communications and Briefing and Display Subsystems.

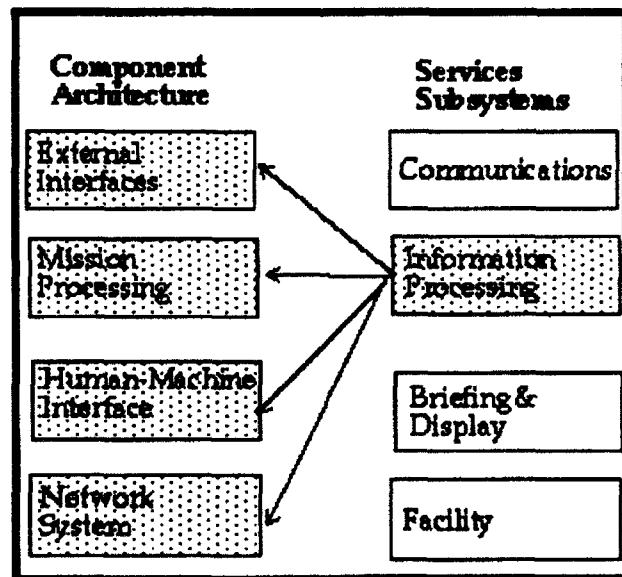


Figure 3-5 Relationship between Component Architecture and Services Subsystems

PRISM's software architecture, upon which the general architecture portion of the library model is based, is shown in Figure 3-6. It is also represented as a graphic image in the CARDS CCL Model. The image is located at the PRISM\_Architecture node, and can be viewed by selecting the "Picture Image" action from the pop-up menu.

Messages, in either ASCII or bit-based format, are routed through the Interprocess Communication component to the Message Translator and Validator (MTV), where they are validated and translated into a 'vanilla' SQL format. The translated, validated messages are submitted to the Database Manager (through the Database Broker) for entry into the database. The Database Manager has the capability to logically separate actual and exercise databases and place information in appropriate tables. The operator interfaces with the GIS through an X Window Interface. The primary function for the GIS is to provide the following capabilities:

- Present basic vector map data
- Zoom and pan
- Display of near real time tracks
- Filter map features (e.g., track, roads, rivers, etc.)
- Calculate and display bearing/range line

The capability to produce tables of information through ad hoc queries is required upon operator command through the X Window Interface. The Table Generator is required to produce SQL commands that are routed through the System Manager to the Database Manager and queried results are routed back to the Table Generator.

Finally, a Briefing System capability is implemented through the X Window Interface. The primary requirements of the Briefing Display Subsystem are to have a hypermedia capability; to be capable of producing charts, graphs, and slide shows; and to interface to the Database Manager using SQL. The latter capability allows for the automatic update of predefined charts as in a typical recurring stand-up briefing. Additionally, the Briefing System is capable of accepting scanned images.

The concepts under subsystem represent large command center subsystems. Concepts under disa\_subsystem are the main hardware/software command center subsystems defined in the DISA CCDH including briefing\_display\_hardware, communications, facility, and ips (information processing subsystem). Concepts under gcc\_group are software subsystems of the ips that were defined by PRISM including human\_machine\_interface, network\_mgt\_subsystem, external\_interface, and mission\_processing. As in Version 3.2 of the library, only these ips subsystems have been modeled in any great detail.

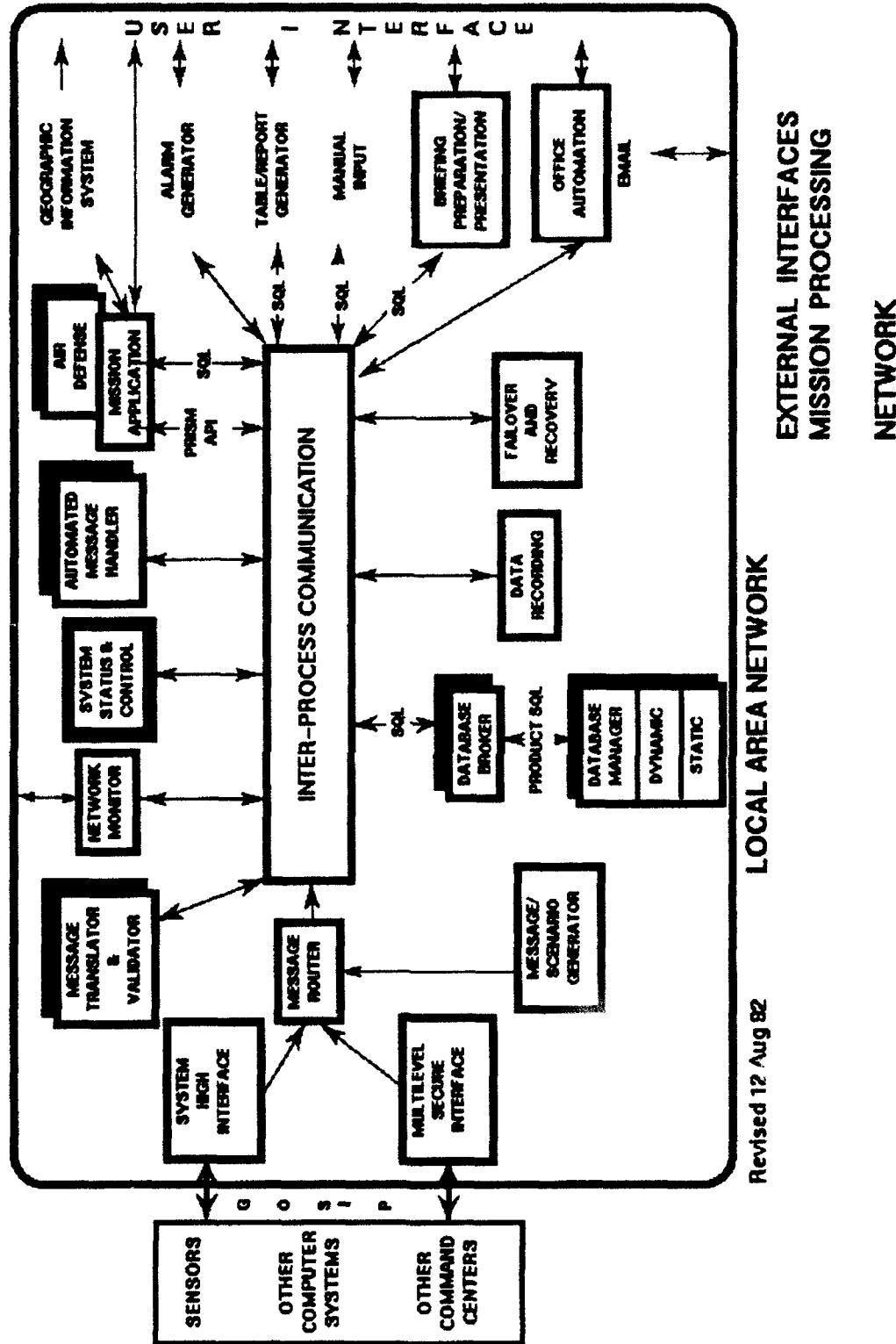
*PRISM Generic Command Center Architecture*

Figure 3-6 PRISM Generic Command Center Architecture

### 3.6 Component Class Models

Each component class is modeled using a standard organization. There are three major classifications for the information: 1) features for that particular component class, 2) architectural constraints and 3) implementation constraints. For each, there is a relationship between the component class and the criteria. The criteria are specified as either categories or objects organized into a semantic network.

#### 3.6.1 Features for that particular component class

These features are functions the component class should perform with respect to a command center. There are features which are required and those which are optional. For example, a feature of a word processor is that it should generate a table of contents.

In the CCL, these criteria can be found under the feature tree. Features which may be common among component classes are normally modeled as direct children of the feature category. Others, more specific to a component class, may have a parent indicating the component class to which they belong. For example, features unique to a spreadsheet exist under a category named, ss\_feature (spreadsheet feature) (Figure 3-7).

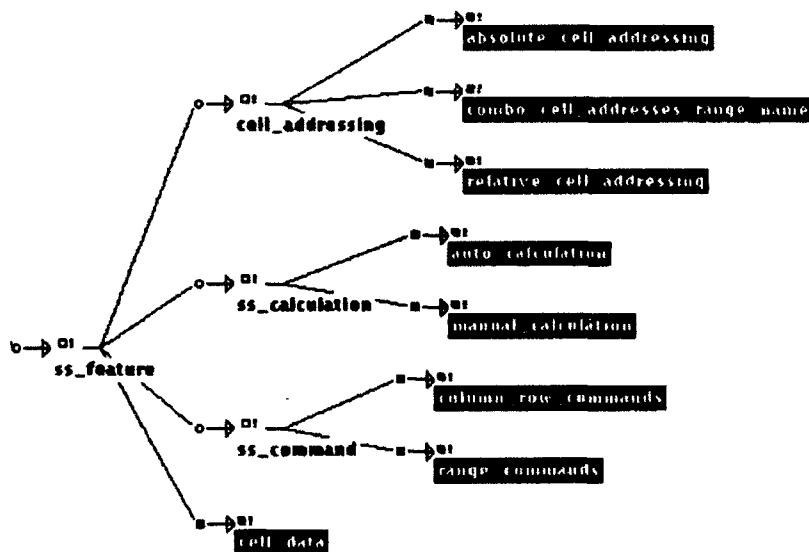


Figure 3-7 Spreadsheet Feature Tree

#### 3.6.2 Architectural constraints

This set of criteria indicates how the component classes fit together to form a command center architecture. For instance, a briefing system must be able to accept data from a word processor. In addition, where required, the criteria indicate how those connections should be implemented. For

instance, a briefing system should be able to accept data from a word processor via interprocess communication.

These criteria are modeled as relationships among component classes. They may appear also as children under the `application_platform_specification` tree (Figure 3-8).

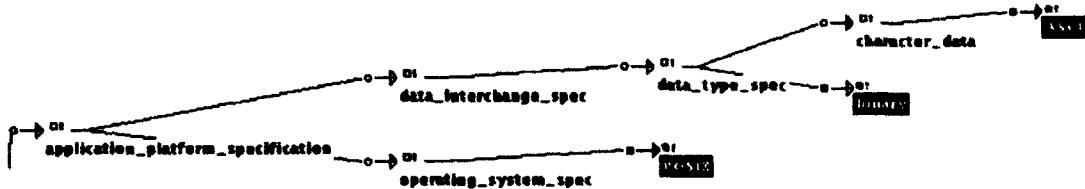


Figure 3-8 Application Platform Specification Tree

### 3.6.3 Implementation constraints

This set of criteria specifies the supporting hardware and software required for the component class. An example of a hardware constraint may be that the component must run on the Sun platform. An example of a software constraint may be that the component must run under X11 Release 4.

In general, hardware constraints are modeled under the hardware tree (Figure 3-9) and software constraints under the `application_platform_software` tree (Figure 3-10).

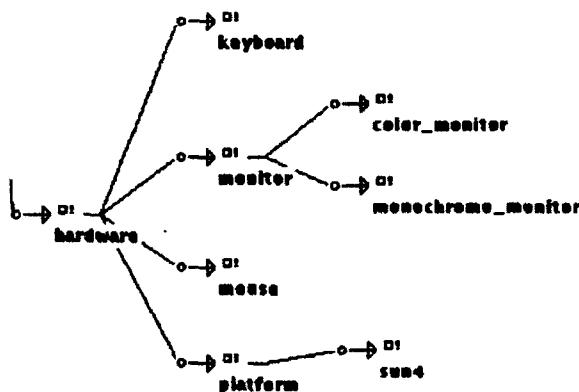


Figure 3-9 Hardware Tree

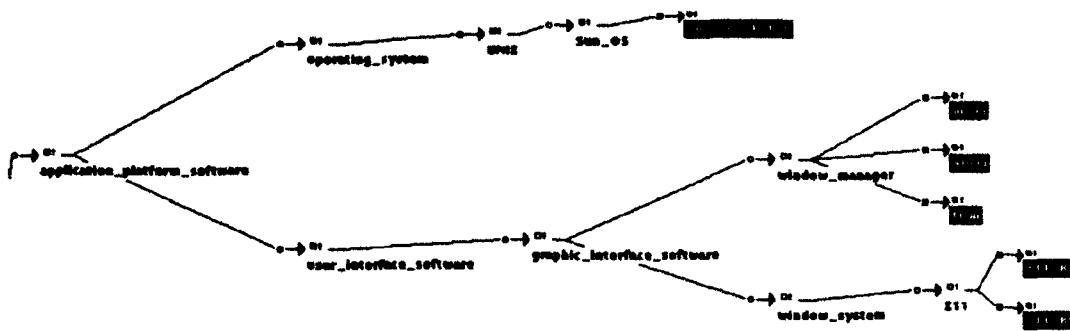


Figure 3-10 Application Platform Software Tree

Features and constraints are either required by the command center (critical features or constraints) or optional to the command center. A critical criterion will have a relationship range of at least one as the minimum [may want to make a reference to the RLF Modeler's Manual Chapter's 3 and 4]. For example, a critical criterion for word processor's with respect to a command center is the ability to generate a table of contents. That would be modeled as:

- does\_generate\_table\_of\_contents (1 .. 1) of table\_of\_contents;

A non-critical criterion will normally have a relationship range of zero as the minimum. For example, a non-critical criterion for word processor's is the ability to create a table. That would be modeled as:

- creates\_table (0 .. 1) of table.

### 3.7 Qualified Components

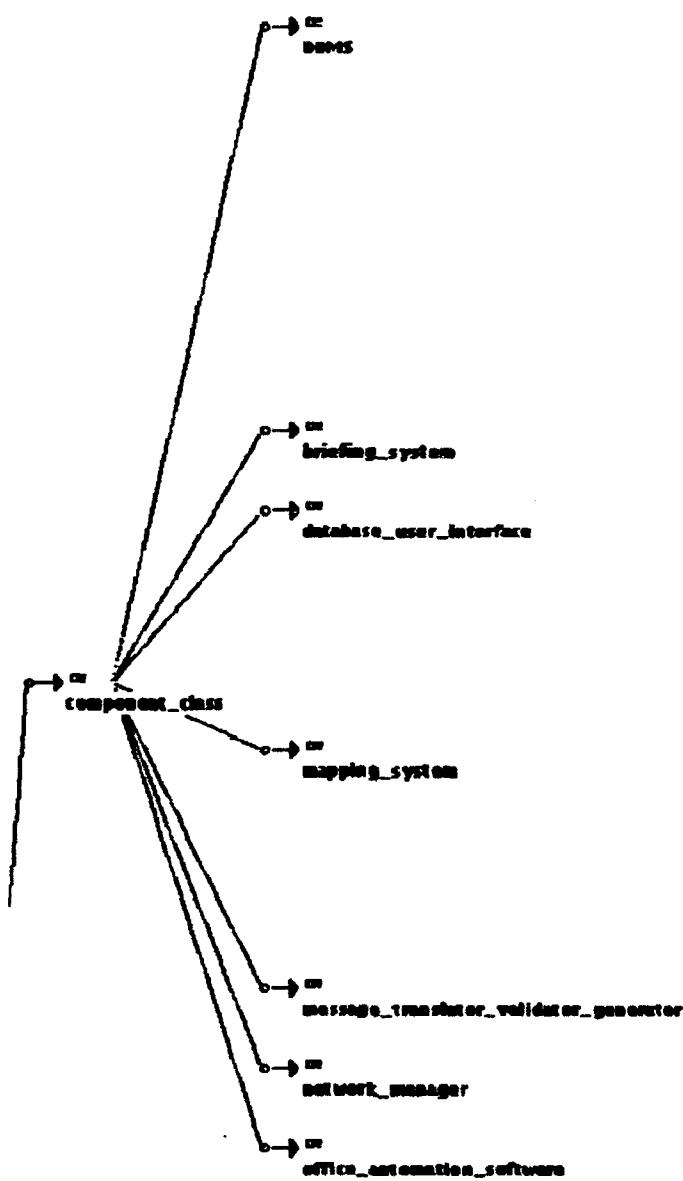


Figure 3-11 component\_class and its children

Component qualification (see Figure 3-7) is the process of acquiring and evaluating components for a domain specific library. The components are qualified as to whether they fit within a particular domain. The emphasis is on supporting domain requirements. The component is classified as to what subsystem of the generic architecture it satisfies. Components are measured against domain criteria, which are measurements of form, fit and function applied to the command center domain. These are a composite of domain, architecture and implementation constraints. Components are also considered by domain independent aspects of common criteria such as performance, reliability, maintainability, etc. After the components have been qualified, they are placed under component\_class category.

### 3.7.1 DBMS

The Database Management System (DBMS) stores and manages tables of data the command center will query and modify during daily activities. The Database Manager should support a relational data model and a client/server architecture. The important and widely used large scale database products now support the client/server architecture.

The DBMS (Figure 3-8) has performance constraints placed upon it by a command center which must be met. An example of a performance requirement is from SAFWCCS: a database manager's response time must not exceed two (2) seconds when performing any single data table transaction.

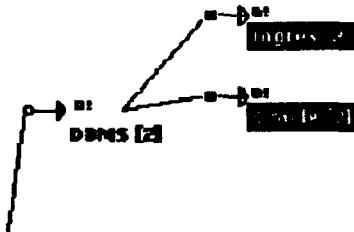


Figure 3-12 DBMS and its children

#### 3.7.1.1 Ingres

Ingres represents release 6.4 of a relational database management system (RDBMS) developed by Ingres Corporation. The RDBMS has been in release since 1983 and about 50,000 licenses have been granted. Ingres provides batch, stored procedures, and interactive Structured Query Language (SQL) data access and manipulation.

Ingres also provides mutual exclusion provisions to provide concurrent data access. The default and lowest level locking for Ingres is page level. Ingres employs locking at various levels; however, the only locks subject to user control are at the table and page levels. Predefined access restrictions are provided to allow application programs to concurrently read and/or write a shared database. Provisions are available for the owner of a database to restrict all or part of the data in the database. Ingres also provides automatic full and partial dump facilities, and the capability for several application programs to concurrently read and/or write the shared database subject only to predefined access restrictions. Provisions are made for the owner of a database to restrict access to all or part of the data in the database.

References to security issues are sparse; however, the vendor claims that Ingres has substantial Command Center (C2) features, and that the next release will have full C2 features. Ingres provides physical and logical separation of data at different security classification levels.

In the CARDS library the following are available for the component Ingres:

- Product evaluation reports in ASCII and postscript formats

- Product assessment on line in model
- Product description in ASCII format

### 3.7.1.2 Oracle

Oracle represents release 7 of a RDBMS developed by Oracle Corporation. The Oracle RDBMS has been in release since 1978 and approximately 400,000 licenses have been granted. Oracle provides support to over 80 platforms.

Oracle7 provides batch, stored procedure, and interactive SQL data access and manipulation, and integrity constraints and triggers as solutions to managing data base data integrity rules. It implements locks to prevent destructive interaction between users accessing the same resource.

Oracle7 and Trusted Oracle (Oracle7 with multi-level security) comply with the National Computer Security Center's C2 and B1 Orange Book Security Criteria levels, respectively. Oracle7 provides extensive auditing, allowing the auditing of data accesses, successes and failures, and logons.

Oracle7 supports client/server operation, a dedicated server architecture in which every user process connected to Oracle7 has a corresponding server process, a multi-threaded architecture in which a small number of shared server processes can perform the same amount of processing that would otherwise be done by many dedicated server processes, distributed processing, location transparency, network transparency, and On-Line Transaction Processing.

In the CARDS library the following is available for the component Oracle:

- Product evaluation reports in ascii and postscript formats
- Product assessment on line in model
- Product description in ascii format

### 3.7.2 Briefing System



Figure 3-13 Briefing\_system and its child

Briefing system (Figure 3-9) is a class of components facilitating the generation of textual and graphical presentations on screen or paper. These components use mission data from command center databases to create bar charts, pie charts and other decision support displays for operations briefings.

### 3.7.2.1 Lotus\_123

Lotus 1-2-3 is a commercial briefing system software package allowing users to create spreadsheets of data and display them as spreadsheets, charts, or graphs. Lotus 1-2-3 offers up to 256 pages of data, built-in functions and 3-D charting capability.

In the CARDS library the following are available for the component Lotus\_123 as a briefing system:

- Product evaluation reports in ASCII and postscript formats
- Product assessment on line in model
- Product description in ASCII format

### 3.7.3 Database User Interface



Figure 3-14 Database\_User\_Interface and its descendants

A database user interface (Figure 3-10) provides a means for the user to interact with the database engine. A typical graphical user interface may provide the user with menus, buttons and/or icons with which to access the database.

### 3.7.4 Database Front-End

The database front-end is used to query the database and produce reports on the database tables. A database front-end provides an interface for the operator of a database to generate structured query language commands. The database front-end should be able to support ad-hoc queries to the database. The database front-end is responsible for the look and feel of the queries to the database. The database front-end also interfaces with the operational database through a database broker.

The database front-end should:

- Provide tools for creating a robust application interface.
- Provide basic transparent data handling, such as automatically creating default forms based on existing tables.

- Simplify menu creation and maintenance.
- Make it easy to incorporate SQL queries in an application.
- Provide a report layout facility that lets the user easily create a report.
- Provide debugging utilities.
- Have satisfactory response times.

### 3.7.5 Multi-Database Database Frontend

A multi-database database front-end can access multiple database management systems.

#### 3.7.5.1 SmartStar

SmartStar Vision is a workstation-based software environment for developing "Motif-to-SQL" database applications with little or no coding. It was developed by SmartStar Corporation and has been in release since November 1992.

The application development environment is very impressive. In addition to the usual Interactive SQL (ISQL), 3rd Generation Language (3GL) library interface, and 4th Generation Language (4GL), SmartStar Vision contains "NoGL." "NoGL" enables a user to create a Graphical User Interface (GUI) databases without SQL, 3GL, or 4GL. It uses menus, buttons, and icons. Once taught about the basics of what a database is, even the most inexperienced user could create effective databases.

SmartStar Vision allowed databases to be developed independent of the physical database type(s) in use. This was accomplished using SmartStar Vision's ANSI-SQL database called Logical Database (LDB). The LDB dictionary stores logical mappings to tables in data files or physical databases.

SmartStar Vision currently supports Oracle, Sybase, and Ingres database management systems. SmartStar Vision was evaluated in conjunction with Sybase.

SmartStar Vision supports both user interface and database client/server connection. SmartStar's GUI client/server support enables any X-compatible device to display SmartStar Vision running on another computer through a remote login.

In the CARDS library the following are available for the component smartstar:

- Product evaluation reports in ASCII and postscript formats
- Product assessment on line in model
- Product description in ASCII format

### 3.7.6 Mapping System

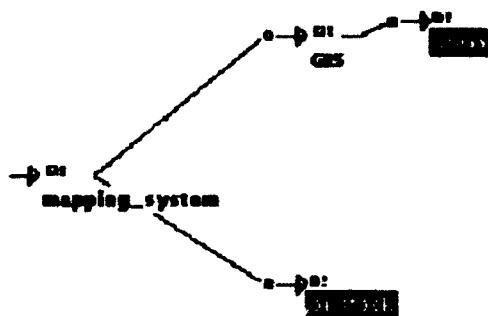


Figure 3-15 mapping\_system and its descendants

A mapping system (Figure 3-11) is a component class designed primarily for the display and manipulation of spatial data and information.

#### 3.7.6.1 OILSTOCK

OILSTOCK is a high resolution interactive graphics system. In addition to data display, analysts can draw geopositional overlays on OILSTOCK maps to produce reports or amplify information. OILSTOCK uses the CIA World Database II (WDBII) as its foundation for vector maps and the Defense Mapping Agency's (DMA) ARC Digitized Raster Graphics for high quality raster maps. OILSTOCK reads Digital Terrain Elevation Data to conduct a visual line of site analysis.

Advanced features include ELINT/Direction Finding capabilities and calculating satellite footprints. OILSTOCK also supports cartographic overlays and numerous map projections. In particular, OILSTOCK is designed to track airborne targets, including aircraft and satellites, and seagoing vessels.

OILSTOCK is accompanied by excellent hard-copy manuals, making it easy to install and evaluate. The organization of the manuals makes them easy to use as a reference and a tutorial.

In the CARDS library the following are available for the component OILSTOCK:

- Product evaluation reports in ASCII and postscript formats
- Product assessment on line in model
- Product description in ASCII format

### 3.7.7 Geographic Information System

A Geographic Information System (GIS) is used for presenting and manipulating information in a geographical context. In basic terms, GIS links tabular data as found in a traditional database

with the visual world of maps and charts. This has proven to be an efficient method of analyzing data and is becoming very wide-spread.

GIS comes in basically two graphic formats: vector and raster. Each type is used to solve different types of problems. Vector data is easily scaled and identified. It is simple to select vector features to be displayed, while ignoring unwanted data. Unfortunately, vector data is very expensive to collect and encode.

Raster data, on the other hand, is inexpensive to produce, since it basically consists of scanned pixels of data. The data looks real, or at least as real as the source of the data. It, too, has its disadvantages, in that it is expensive to store and virtually impossible to update without adding rescanned images. The ability to support both vector and raster processing has greatly increased the usefulness of GIS and has fueled its wide-spread use and increased availability in recent years.

A GIS should provide the ability to add well-defined standard symbols to 2- and 3-dimensional maps, and provide tools to perform analysis on both the maps and added symbols. Typical analysis requires polygon overlay, network routing, address geocoding, and spatial statistics. Communicating with a database is necessary to add dynamic and static data to the view of the map.

Many GISs can also interface with output from computer aided design (CAD) systems and existing tabular data.

### 3.7.7.1 GRASS

GRASS is a public domain GIS that is popular within both the government and academic communities. It was originally developed by the Construction Engineering Research Laboratory of the United States Army. Originally designed for resource analysis, it has since been expanded and extended to perform a wide variety of applications involving the display, manipulation, and analysis of spatial data.

The primary advantages of GRASS are its display and analysis capabilities; and its ability to be extended and used in a variety of methods. GRASS is comprised of two modules: a C function library and a stand-alone system. Both provide a large degree of functionality and can be extended either through a client application with the function library or by developing custom commands via the stand-alone environment.

The flexibility of GRASS makes it an excellent addition to a CCL because of the diverse and critical requirements of a command center.

In the CARDS library the following are available for the component grass:

- Product evaluation reports in ASCII and postscript formats
- Product assessment on line in model
- Product description in ASCII format

- Product source code

### 3.7.8 Message Translator/Validator Generator

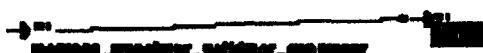


Figure 3-16 message\_translater\_validator\_generator and its child

The message translator/validator generator (Figure 3-12) is a component class that produces other components. This particular node produces or generates systems that perform the tasks of message translators and validators. Typically generators produce source code and it is then the responsibility of the application developer to compile the code and perform any necessary integration.

#### 3.7.8.1 GBMV

The GCCA specifies a component for message translation and validation (MTV), for the processing of encoded messages. The Grammar Based Message Validator (GBMV) provides the validation functionality, creating validators capable of processing character-based encoded messages. Users of the GBMV can produce validators for a wide range of character-based message formats, with little training and a basic knowledge of regular expressions.

The GBMV runs under UNIX and incorporates the Ada-based compiler generation tools Aflex and Ayacc, developed at the University of California - Irvine. An Ada compiler is required for the compilation of these tools, and for the compilation of the validator code generated by the GBMV.

In the CARDS library the following are available for the component GBMV:

- Product evaluation reports in ASCII and postscript formats
- Product assessment on line in model
- Product description in ASCII format
- Product source code
- Product test executable code
- Product user manual in postscript format

### 3.7.9 Network Manager

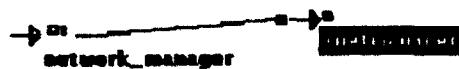


Figure 3-17 network\_manager and its child

A network manager (Figure 3-13) is a component class designed primarily for the tracking, management and control over physical and local network resources connected together via a LAN.

#### 3.7.9.1 XNetManager

XNetManager is a package of public domain software components and enhancements to support the five major functional areas of network management required by the GCCA: fault management, configuration management, accounting management, performance management, and security management. XNetManager v1.0 is comprised of three public domain packages: Xnetdb-v2.10 from the Ohio State University, XNetmon-v1.0 from the Delft University of Technology of Holland and xnetmon-v1.1 from the University of Wisconsin.

XNetManager has proven to be functionally adequate for a portion of network management for a GCCA based on the requirements set forth by the *domain criteria*. XNetManager provides many basic functions of network management, specifically in the areas of fault and performance management, and is deemed an adequate tool for assisting the network operation manager or system administrator in getting a handle on an operational local area network.

### 3.7.10 Office Automation Software

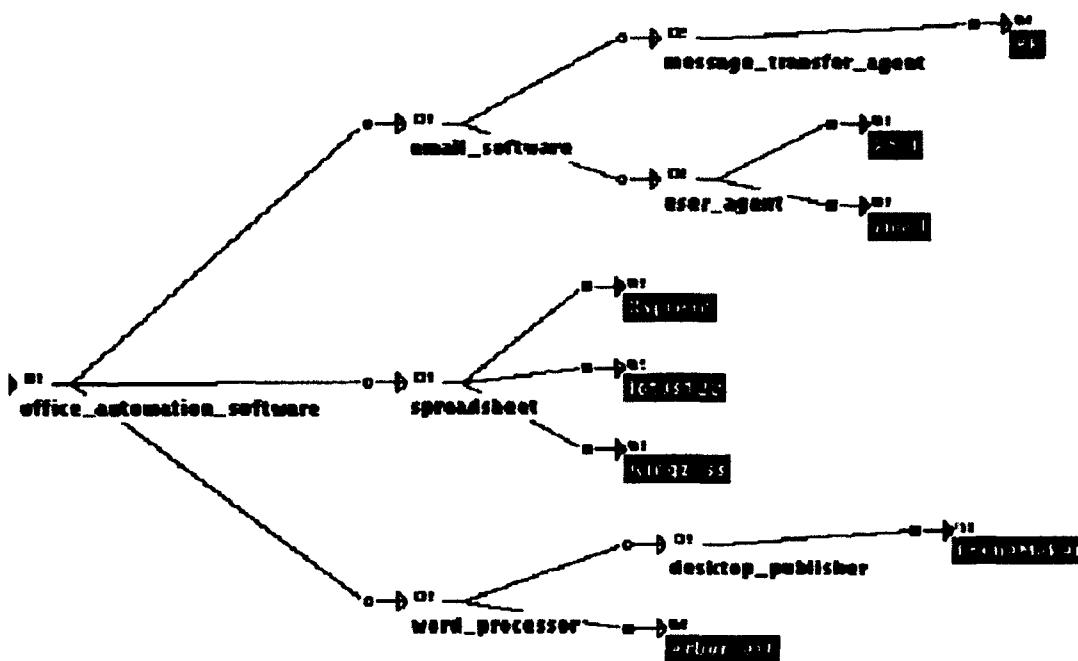


Figure 3-18 office\_automation\_software and its descendants

Office Automation (Figure 3-14) is an augmentation to the main mission support capabilities and includes applications such as word processing, spreadsheets, and graphics generation. It is able to interface to the database and other internal management information systems. This component may be used in conjunction with the briefing preparation/presentation or other mission operations. It also provides electronic transfer of informal user messages e-mail over prescribed LAN segments. It includes functions such as mail storage, retrieval, mail broadcast, etc.

### 3.7.11 Electronic Mail Software

Electronic mail (e-mail) is a required office automation function. It permits routing of incoming mail based on controls established by supervisory personnel using plain English addresses. The e-mail component must be able to support different file types from within the same message. This includes ASCII, graphics, scanned images, FAX, spreadsheets, CAD drawings, and binary files. Other functions includes the capability to:

- Designate different user classes and access levels.
- Locate stored messages by sender, receiver, keyword search, or text string.
- Set up bulletin boards, LAN-wide aliases, and distribution lists.

There are two basic sub-processes of e-mail that implement these various capabilities:

1. Message Transfer Agent (MTA) - This process is responsible for the actual transmission and reception of mail messages across the network (both internal and external).
2. User Agent Process (UAP or UA) - This process is the front-end or user interface to the MTA.

### **3.7.12 Message Transfer Agent**

An e-mail MTA is responsible for transporting mail from sources to destinations, possibly transforming protocols and addresses, and routing the mail to the appropriate recipients.

The MTA often has several components:

- Routing mechanisms
- Local delivery agent
- Remote delivery agent

Many MTA's have all of these components. In other cases, it is possible to replace certain components for increased functionality.

#### **3.7.12.1 PP**

PP is a public domain message transfer agent in the e-mail component class, intended for high volume message switching, protocol conversion, and format conversion. PP supports the 1984 and 1988 versions of the CCITT X.400 services and protocols. Many existing RFC 822 based protocols are also supported, along with RFC 1148 conversion to X.400.

PP is an appropriate replacement for traditional UNIX MTAs such as MMDF (Multichannel Memo Distribution Facility) or sendmail.

With a clean interface for message submission and delivery (e.g., support for both MMDF and sendmail style mailboxes), most widely available MTAs work in conjunction with PP to provide a fully functional e-mail system. It was specifically tested and worked well with Z-Mail (see section 3.6.13.1), MH (with and without the metamail MIME patches), XMH (with and without the metamail MIME patches), sendmail, elm, xmail, and OpenWindows mailtool.

In the CARDS library the following are available for the component PP:

- Product evaluation reports in ASCII and postscript formats
- Product assessment on line in model

- Product description in ASCII format
- Product source code

### 3.7.13 User Agent

An e-mail user agent (UA) is the user interface to the e-mail system and minimally provides the functionality required to view incoming mail and compose outgoing messages. In support of these functions, the UA must be able to delete, print, store, reply to, and forward mail messages.

User agents typically have additional support for mail manipulation, including sorting, folder operations, and keyword searches of message content. More advanced features include reading/composing mail messages containing arbitrary/non-text data (attachments), modifying headers, notifying the sender when the receiver has read the mail, etc.

#### 3.7.13.1 zmail

Z-Mail is a UA of the e-mail class of components maintained by Z-Code Software Corporation in San Rafael, California.

Z-Mail provides extensive message composition capabilities, including the ability to include file formats such as binary, PostScript, bitmap, fax, and audio with a mail message. Z-Mail also provides useful message searching and sorting capabilities: users can search for patterns within the messages or search for messages based on single or multiple expressions. Actions can then be taken on those messages matching the expression(s). Messages can also be sorted in ascending or descending order on multiple keys.

Z-Mail offers a scripting language, Z-Script, for creating message filters and other functions that help to automate management of mail. Z-Script allows a user to create custom defined buttons, user defined functions, and attach Z-Script to a mail message.

Some other important features of Z-Mail include: the ability to perform any action on a group (tagged) of messages, on-line help, the ability to issue shell commands from within the mailer, notification to the sender that a message has been received, font editing, and color editing.

In the CARDS library the following are available for the component zmail:

- Product evaluation reports in ASCII and postscript formats
- Product assessment on line in model
- Product description in ASCII format

#### 3.7.13.2 XMH

XMH is an e-mail UA whose basic configuration consists of two public domain packages, MH and xmh. MH is a text-based user agent, while xmh provides an X-based user interface to MH.

MH differs from typical UAs in that all of its commands are separate executables, providing the ability to execute MH commands from UNIX shell programs and source code. xmh provides graphical user interface (GUI) functionality by making calls to the MH programs.

Two other packages, metamail and xlbiiff, have been identified for use in conjunction with XMH to provide required functionality. Metamail converts XMH into a MIME (Multipurpose Internet Mail Enhancements) compliant multimedia UA (although message creation is supported only in the text mode). xlbiiff alerts the user of new mail.

Standard user agent functionality is supported, including: viewing, creating, forwarding, replying to, printing and deleting mail; folder operations (creating, opening, saving mail to), and tagging messages for group operations. XMH also supports sequences - associating a group of messages with some name based on characteristics of those messages.

Metamail provides the ability to include arbitrary, non-text data in a mail message. A message can be separated into multiple parts, and transmitted as separate mail messages. Another notable feature is that of including "pointers" to data instead of the data itself (e.g., a file at a remote site). metamail will retrieve the data when the message is read.

As for architectural constraints, XMH has direct MTA support for MMDF (Multichannel Memo Distribution Facility), MMDF-II, sendmail, and zmailer. MTAs providing interfaces based on these four transfer agents will also work with XMH.

In the CARDS library the following are available for the component XMH:

- Product evaluation reports in ASCII and postscript formats
- Product assessment on line in model
- Product description in ASCII format
- Product source code

### **3.7.14 Spreadsheet**

Spreadsheet is a component class facilitating the entry of tables of numbers. Spreadsheets allow users to specify equations that calculate other table entries based on the user input. Spreadsheets often allow users to create bar charts, pie charts and other graphical presentations of selected data from the numerical tables.

#### **3.7.14.1 Xspread**

Xspread is a public domain spreadsheet based on the spreadsheet driver program SC. It was developed at the University of Wisconsin at Milwaukee. There is help available, documentation, source code and an ftp-site listing on-line.

In the CARDS library the following are available for the component xspread:

- Product evaluation reports in ASCII and postscript formats
- Product assessment on line in model
- Product description in ASCII format
- Product source code

### **3.7.14.2 lotus123 (spreadsheet)**

lotus123 is a commercial spreadsheet software package. This package allows the user to create spreadsheets of data and display them as spreadsheets, charts, or graphs. lotus123 offers up to 256 pages of data, built-in functions and 3-D charting capability.

In the CARDS library the following are available for the component lotus123 as a spreadsheet:

- Product evaluation reports in ASCII and postscript formats
- Product assessment on line in model
- Product description in ASCII format

### **3.7.14.3 wingz\_ss**

Wingz is a commercial spreadsheet package. It has very extensive graphic capabilities with 3-D imaging available as well as a Hyperscript language to enhance user documents. Wingz is available for all the major graphical interfaces, making it very portable and usable.

In the CARDS library the following are available for the component wingz\_ss:

- Product evaluation reports in ASCII and postscript formats
- Product assessment on line in model
- Product description in ASCII format

## **3.7.15 Word Processor**

A word processor allows for the creation and modification of textual documents. Capabilities normally include importation of graphics, setting of font styles and sizes, creation/modification of paragraph formats, etc. The most common form of interacting with a word processor is through a WYSIWYG (What You See Is What You Get) style of graphic user interface. Many common

formats exist for storing documents created with word processors. These include: PostScript, Standard Generalized Markup Language (SGML), EDI, and other vendor specific formats along with ASCII.

### **3.7.15.1 Arbortext**

The Arbortext ADEPT Series provides a structured approach to documentation preparation. The ADEPT Series includes two products aimed at structured document preparation:

- SGML Editor - a text editor for creating and editing SGML documents.
- SGML Publisher - provides the same tools as the SGML Editor, but adds previewing and printing capabilities.

In the CARDS library the following are available for the component arbortext:

- Product evaluation reports in ASCII and postscript formats
- Product assessment on line in model
- Product description in ASCII format

### **3.7.16 Desktop Publisher**

Desktop publishing systems are similar in capabilities to word processors but with advanced features. Some of these advanced features include: the capability to create/edit graphic figures and images within the system, and functionality for specifying the layout of the document.

### **3.7.16.1 FrameMaker**

FrameMaker is a Commercial Off The Shelf (COTS) component facilitating WYSIWYG creation of formatted reports with embedded graphics. FrameMaker is an advanced publishing tool that integrates word processing, graphics, page layout, and book building. It also allows users to import various document and graphics formats.

In the CARDS library the following are available for the component FrameMaker:

- Product evaluation reports in ASCII and postscript formats
- Product assessment on line in model
- Product description in ASCII format

### 3.8 Reference Model

In the structural, AdaKNET portion of the model, the model closely reflects the generic architecture established by PRISM along with requirements information from the DISA CCDH [DISA91]. What has changed is the way hardware and software components, that provide services to higher level application software, are organized. This new organization, called an *application platform*, was derived from the PRISM Technical Reference Model (Figure 3-15) which is based on the NIST Applications Portability Profile [NIST] and the DoD Technical Reference Model [DOD92].

The graphic image of the PRISM Technical Reference Model is also available for viewing in the CARDS CCL Model. The image is located at the PRISM\_TRM node and is produced by selecting the "Picture Image" action.

The concepts under *application\_platform\_entity* (Figure 3-16) provide services to higher level application software. The application platform or technical reference model forms a basis for defining an architecture permitting interchangeability of components within subsystems. One main category of the application platform is *application\_platform\_specification* which includes standards for operating systems, networking, text and graphics formats, and user interfaces, etc. The other category, *application\_platform\_software* consists of software components which implement the standards and specifications.



### PRISM REFERENCE MODEL

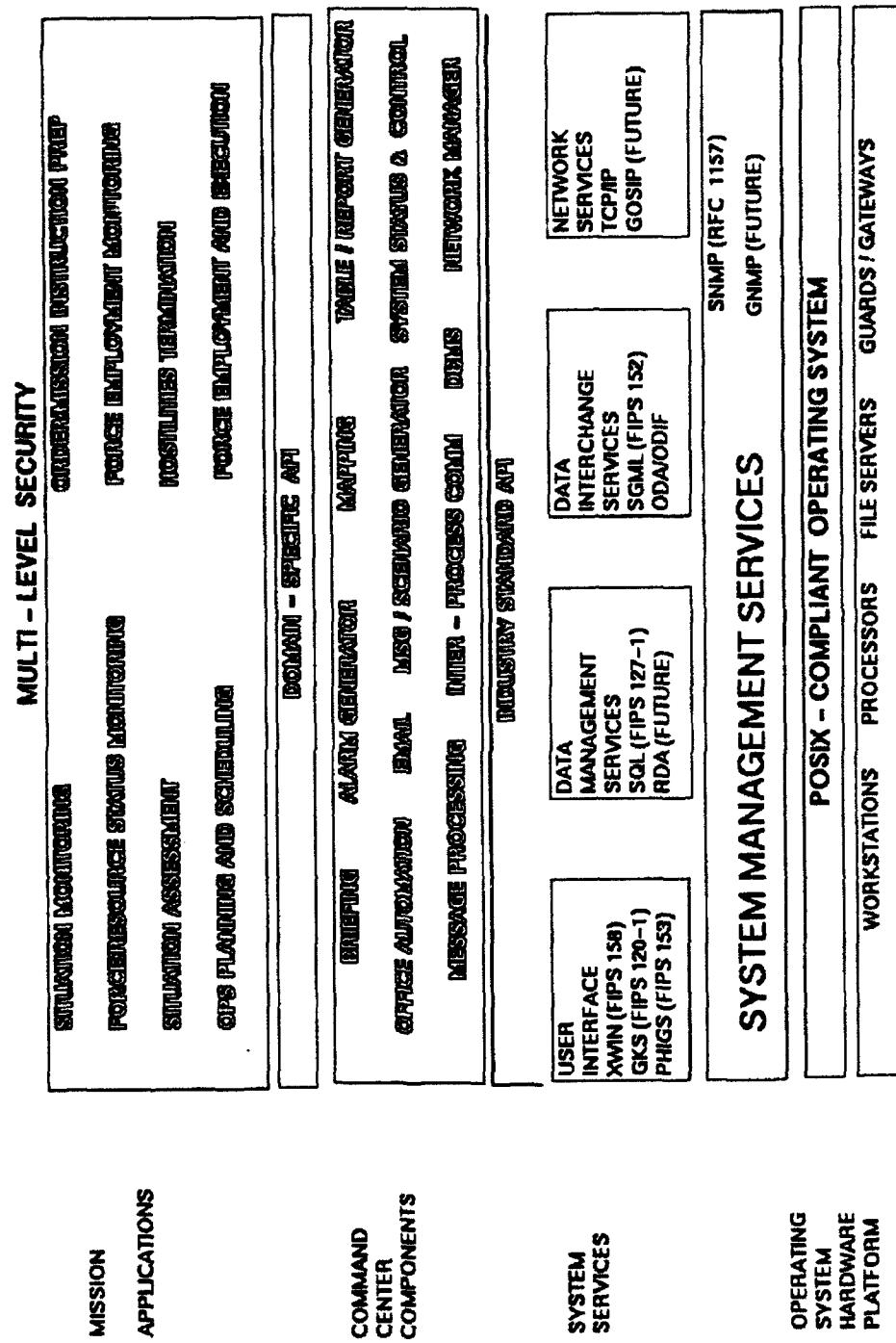


Figure 3-19 PRISM Technical Reference Model



Figure 3-20 application\_platform\_entity and its descendants

### 3.9 System Composition

One of the main purposes of a domain model is to capture information that will help a software engineer build a system. In CARDS, the capability to interactively build a system from reusable components by capitalizing on information in the domain model is called *system composition*. The approach used in the CARDS system composition tool is a form of *knowledge-based software engineering*. Rules for composing a system are written in CLIPS [GIAR92]. The system

composition tool uses these rules to query AdaKNET. It infers pertinent questions to ask the user. It uses the answers to compose the chosen system.

The CARDS system composition tool performs system composition on the message processing subsystem portion of Release 3.3 of the library, and is invocable directly from the library. The following discussion describes the basic operation of this tool:

- The tool elicits information from the user as to what system functionality is required. These questions are based on information about message processing components stored in the library and the relationships among the components represented in the library model. Each answer triggers the selection of appropriate components for the customized message processing subsystem being created by the tool.
- System composition follows the basic process:

**Input:**

- CARDS library model.
- User knowledge of desired system.

**Output:**

- Appropriate source and/or executable code extracted from the reuse library for the desired system.

**Processing Sequence:**

1. User clicks on **Perform Action** at a buildable concept node (i.e., system/subsystem). Note that some nodes cannot be composed into a system because they cannot function as a stand-alone entity.
2. CLIPS queries AdaKNET to retrieve roles that must be filled. For example, a PRISM message generator subsystem has roles for user interface and message formats. CLIPS determines from the model what versions of components are available that implement the user interface and message formats.
3. CLIPS asks the user questions that allow all necessary roles to be filled.
4. CLIPS asks domain dependent questions relevant to the particular system being composed.
5. If the preconditions for building the chosen node are met (i.e., all questions are answered), CLIPS then *harvests* the system by collecting all the role filler information.

It then generates calls to shell scripts which put the source/executable code together in user specified directories.

The following is a simple example of system composition from an actual transcript for a PRISM message generator subsystem. A 'help' file containing detailed information about the operation of the tool is displayed when the tool is initially invoked.

User input is in numeric form and appears after the arrow symbol '*->*'. Commentary on the transcript appears in *italic* typeface.

\*\*\*\*\*  
\* WELCOME TO THE SYSTEM COMPOSITION DEMO \*  
\*\*\*\*\*

\* (Press ENTER to continue) \*

\*\*\*\*\*  
\*\*\*\*\*  
Choose the number corresponding to  
your choice below, then press ENTER:  
\*\*\*\*\*

1. Build the current component: message\_processing  
(traverse the command\_center LMDL network)
2. Demo/execute the message\_processing code
3. Exit (return to RLF)

\*\*\*\*\*  
->1  
\*\*\*\*\*  
\*\*\*\*\*

\* Invoking network traversal to build the \*  
message\_processing concept  
(Press ENTER to continue)

\*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*  
Enter the directory where you would like to store

the message\_processing component, then press ENTER.

\*\*\*\*\* NOTE \*\*\*\*\*

\*\*\* the directory entered must be a valid \*\*\*

\*\*\*directory with write access. \*\*\*

->/my-dir

Which of the following do you want to include  
for your message\_processing component:

Source code only

Executable(s) only

Both source code AND executable(s)

->1

THE CURRENT CONCEPT message\_processing

(has role) translates\_validates

(of type) message\_translator\_validator

Select one of the following - if no choice  
user choice is desired, choose the number for 'None',  
then press ENTER

(type 'exit' to return to RLF)

1. MTV

2. None

->1

## THE CURRENT CONCEPT MTV

(has role) processes

(of type) `gcc_message_format`

\*\*\*\*\*

Select one or more of the following,  
then press ENTER

(type 'exit' to return to RLF)

\*\*\*\*\*

1. `e3a`
2. `intercept`
3. `nudet`
4. `roccsocc`
5. `unsecure_tadil`

->2

\*\*\*\*\*

## THE CURRENT CONCEPT message\_processing

(has role) `message_source`

(of type) `message_generator`

\*\*\*\*\*

Select one or more of the following,  
then press ENTER

(type 'exit' to return to RLF)

\*\*\*\*\*

1.  
`ascii_prism_msg_gen`  
`bb_prism_msg_gen`

->1

Note: Interaction omitted where user is queried for  
the interface\_type of ascii\_prism\_msg\_gen, the X11 release  
to use, and the type of message format that the ascii\_prism\_msg\_gen  
should generate.

\*\*\*\*\*

### THE CURRENT CONCEPT message\_processing

(has role) component\_communication

(of type) interprocess\_communication

Select one of the following - if no choice is desired,  
choose the number for 'None',  
then press ENTER

(type 'exit' to return to RLF)

- 1. dec\_message\_q
- 2. plain\_sys\_man
- 3. sys\_man\_ingres
- 4. sys\_man\_sybase
- 5. None

->2

\*\*\*\*\*  
\*\*\*\*\*

END OF NETWORK TRAVERSAL ALGORITHM

READY TO INVOKE DOMAIN-DEPENDENT QUESTIONS -

PRESS ENTER TO CONTINUE

\*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\* ABOUT TO HARVEST \*\*\*\*\*

ASCII PRISM MESSAGE GENERATOR

PRESS ENTER TO CONTINUE

\*\*\*\*\*

\*\*\*\*\*

COMPONENT RETRIEVAL Date: Thu Mar 25 09:06:37 EST 1993

COMPONENT SELECTION:

Component: ascii\_prism\_msg\_gen\_motif -

ASCII PRISM Message Generator

Message Mix: 60,0,20,20

Windowing System: Motif Graphical User Interface

Target Directory: /my-dir/ascii\_prism\_msg\_gen\_motif

Packaging Content: Source Code Only

Extracting ascii\_prism\_msg\_gen\_motif/src . . . Done.

COMPONENT RETRIEVAL COMPLETED

Press RETURN to EXIT!

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\* ABOUT TO HARVEST \*\*\*\*\*

PLAIN SYSTEM MANAGER

PRESS ENTER TO CONTINUE

\*\*\*\*\*

\*\*\*\*\*

COMPONENT RETRIEVAL Date: Thu Mar 25 09:07:11 EST 1993

COMPONENT SELECTION:

Component: plain\_sys\_man -

PRISM System Manager without MTV.

Target Directory: /my-dir/plain\_sys\_man

Packaging Content: Source Code Only

Extracting plain\_sys\_man/src . . . Done.

COMPONENT RETRIEVAL COMPLETED

Press <return> to EXIT!

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\* ABOUT TO HARVEST \*\*\*\*\*

MTV

PRESS ENTER TO CONTINUE

\*\*\*\*\*

\*\*\*\*\*

COMPONENT RETRIEVAL Date: Thu Mar 25 09:07:32 EST 1993

COMPONENT SELECTION:

Component: mtv - PRISM Message Translation and

1. Validation (MTV) Standalone.

Target Directory: /my-dir/mtv

Packaging Content: Source Code Only

Extracting mtv/src . . . Done.

COMPONENT RETRIEVAL COMPLETED

Press <return> to EXIT!

Configuration completed.

Press <return> to continue.

### 3.10 Component Qualification Tool

The Component Qualification Tool provides our users a method of qualifying potential components within the command center domain. The tool also provides staff software engineers the ability to qualify potential components for inclusion in the CARDS CCL Model.

Each component class has relationships, or attributes, that a component can potentially fill. These relationships describe how a component fits within a particular domain such as the command center domain. To qualify a component, a component is evaluated against the relationship of the component class. This relationship is of two types: critical and non-critical. The critical relationships are the relationships that must be met for the component to be "qualified" into the component class. The non-critical relationships are ones that are found frequently within the domain of interest, but a component is not required to have. For example, for a command center domain in the component classes of office automation software, a critical relationship could be that the software must be able to input ascii form text, but a non-critical relationship could be that the component could have a spellchecker. If a component does not meet all critical roles of a component class, it cannot be considered "qualified." If a component meets all critical relationships, but not all of the non-critical relationships, it is "qualified" for that particular component class.

By querying the RLF network, the component qualification tool finds the critical and non-critical roles of the component class. It then prompts the user with questions pertaining to the relationship of the component class that they are qualifying against. When all the questions are answered, the tool then generates a report and tells the user if the component has "passed" or "failed" qualification. If the component has passed, the RLF LMDL code is generated for inclusion into the library at a later time.

Components which provide the action, "Qualify Component", are included in Appendix C, Library Actions.

### **3.11 System Demonstrations**

The CARDS CCL Model provides users the ability to evaluate software components through demonstrations. In some cases the demonstrations will be full working versions of the software with no limitations; other cases will be demonstration versions of the software supplied by the vendor or a "screen capture" walk-through of the software.

#### **3.11.1 ascii\_PRISM\_msg\_gen**

The ascii\_PRISM\_msg\_gen component generates and injects simulated ASCII messages into the Generic Command Center (GCC) IPS. It is configured by the command center operator through the user interface and injects messages into the GCC IPS through the external interfaces component.

#### **3.11.2 bb\_PRISM\_msg\_gen**

The bb\_PRISM\_msg\_gen component generates and injects simulated bit-based messages into the GCC IPS. It is configured by the command center operator through the user interface and injects messages into the GCC IPS through the external interfaces component.

### **3.12 Interoperability**

In CCL release 3.2, CARDS implemented trilateral interoperability, i.e., interoperation between CARDS, ASSET, and DSRS. CARDS library interoperability uses a client/server architecture to retrieve components from a remote library. When a CARDS user requests an abstract, description, or an entire component residing at another library, a program is run which connects to a central CARDS interoperability server running on the CARDS host machine in Fairmont, WV. This server then verifies the credentials of the user, and connects to the remote library to retrieve the component files. The files are retrieved over the Internet using TCP/IP.

The user does not need to do anything special to work with components residing at another library. Everything is handled for them.

The CARDS library includes components from two remote libraries - DSRS in Falls Church, VA, and ASSET in Morgantown, WV.

#### **3.12.1 DSRS Interoperability Components**

The following four interoperability components were entered into the CARDS Command Center domain for Release 3.2, but actually exist within the DSRS Library. These components are modeled as individuals in our library, but the actual source and binary files reside in the DSRS Library. Actions exist at the individual nodes for Display Abstract and Extract Contents.

##### **3.12.1.1 Screen\_And\_Data\_Manager\_Package**

Screen\_And\_Data\_Manager\_Package is an Ada package that provides 45 routines for: (1) managing a screen definition; (2) displaying data to a user at a terminal; and (3) retrieving data from a user at a terminal. All of the routines in the package operate on variables of types defined in the package specification. A screen is defined to be an array of records, where each record defines a field on the screen. Screen\_And\_Data\_Manager\_Package is the child of ity\_interface\_software.

Reuse of this package would be valuable for a system required to maintain, modify, and query a screen definition, as well as to display and retrieve data to/from a user at a terminal, provided an implementation of Terminal\_Interface\_Package is available.

##### **3.12.1.2 Generic\_Report\_Handler**

Generic\_Report\_Handler package was designed to be a package that facilitates the generation of printed reports. The user supplies procedures that handle the generation of column headers, footers, etc., and the report handler calls upon user-supplied procedures as necessary. The primary advantage of using the package lies in the "virtual page" abstraction that frees the user from the burden of ensuring that page feeds are done at the right time. Generic\_Report\_Handler is the child of ity\_interface\_software.

### 3.12.1.3 Safe\_IO

Safe\_IO is a package that allows the user to input data types from the keyboard while checking the input for errors. A procedure for checking input of characters for a proper sub-range of the character set is provided. When an error is encountered, an error message is displayed and the user is allowed to re-enter. Output routines are provided to allow the user to do I/O with only one instantiation. Safe\_IO is the child of *itty\_interface\_software*.

### 3.12.1.4 String\_Utils\_Package

String\_Utils\_Package is an Ada package that provides operations for parsing and maintaining strings and text. The subprograms in this package provide the user with the capability to input an entire string or just a segment of the string without losing the index position within the original string. The largest string length this package can support is system dependent, i.e., the largest positive integer supported by the system. String\_Utils\_Package is the child of *itty\_interface\_software*.

## 3.12.2 ASSET Interoperability Components

The following three interoperability components were entered into the CARDS Command Center domain for Release 3.2, but actually exist within the ASSET Library. These components are modeled as individuals in our library, but the actual source and binary files reside in the ASSET Library. Actions exist at the individual nodes for: Display Abstract, Display Relationships Graphically, Display Relationships Textually, and Extract Contents.

### 3.12.2.1 Ada\_SQL\_bindings

A standard binding between Ada and SQL (Structured Query Language), the ANSI and DoD standard for accessing commercial relational database management systems (DBMSs). SQL was not designed to be embedded within applications in general-purpose programming languages, such as Ada. Previously developed Ada-SQL bindings have had various technical drawbacks. Ada\_SQL\_bindings is the child of *language\_bindings\_software*.

A prototype Ada-SQL binding was built by automating the SQL Ada Module Extension methodology (SAME). SAME is a method for building Ada applications that access DBMSs via SQL. SAME extends SQL by exploiting the features of Ada.

### 3.12.2.2 Optimization\_and\_Planning\_Tools

This package provides tools for mission planning. There are general algorithms lending themselves to generic reusable software packages, including the SCT terrain masking algorithm, the two-dimensional implicit-stage multi-pass dynamic programming algorithm, the two-dimensional retrieval algorithm, the three-dimensional extensions of the MDPA and route retrieval algorithm, and the Dijkstra shortest path algorithm. Optimization\_and\_Planning\_Tools is the child of *component\_class*.

### 3.12.2.3 Reusable\_Image\_Processing\_Package

The Reusable Image Processing Package is a set of reusable Ada packages performing various image processing functions, including image enhancing and image statistic modeling. *Reusable\_Image\_Processing\_Package* is the child of *component\_class*.

### 3.12.3 Interoperability Metrics

During the implementation of trilateral interoperability, staff developers added a feature to the CARDS Library to automate the collection of interoperability metrics. The process was enlarged to include all components in the CARDS Library, both local and remote.

Interoperability metrics collection is transparent to the user. The metrics collection process is invoked by the CARDS Library actions: Extract Contents, Display Abstract, and Provide Description. The following information is logged into a system file:

- date
- time
- user account name
- library origin (e.g., CARDS, DSRS, ASSET)
- type of transaction (extract, display, or provide)
- success/failure
- transaction time
- component acted upon

## 3.13 Future Directions/Enhancements

### 3.13.1 Structural Changes

The main thrust of future development of the CCL Model will be to continue adding to the knowledge contained in the model to support system composition and other tools.

Currently the representations of software architecture details in the model are not fully "fleshed out". Future plans call for developing improved representations for software architecture within the model, possibly supported by other tools. These improved representations will then be used to fill out the software architecture part of the model.

### **3.13.2 Action Changes**

The Qualify Component action, which allows the user to see or obtain a printout of the optional and required features of the current category, will be added to more nodes in future releases of the library.

A Feedback action allowing the library user to send feedback to the CARDS hotline will be added to the library. The Obtain Help action will be added to more nodes.

Each of the remote actions will be merged with their local counterparts. The intention is that the user will see the same interface, regardless of whether the component is stored in the CARDS Library or another library with whom CARDS interoperates.

### **3.13.3 Anticipated Qualified Components**

#### **3.13.3.1 UNAS\_SALE (Universal Network Architecture Service/Software Architect's Life-cycle Environment)**

There are three product components that come with this product:

- UNAS development kit: a suite of portable, reusable building blocks (a library of pre-existing "primitive parts"), tools, and services. UNAS primitives represent a very high level language for architecting a distributed software system.
- Software Architect's Lifecycle Environment (SALE): a graphical design environment for designing systems out of UNAS parts. The environment is knowledge-based, where UNAS design rules and performance characteristics constitute the knowledge base. The environment generates source code for an architecture skeleton; essentially, it provides the equivalent of a UNAS compiler translating the graphical and textual UNAS source code for any UNAS supported platform.
- UNAS Runtime Kit: a runtime library of scalable components for controlling, instrumenting, monitoring, debugging, tuning and reconfiguring a network of UNAS objects.

CARDS will attempt a two step approach to implementing UNAS and SALE into the CARDS Library:

1. Placing it into an existing component class defined within the GCCA. Currently, it is placed within the (unqualified) CASE\_tools component class; however, it may be later qualified under another class. Possible component classes include interprocess communications, system status and control, and/or network monitoring.
2. Using UNAS and SALE in an ongoing CARDS survey of software architecture concepts and technologies. The goal of this longer-range activity is to capitalize on the

software architecture/survey to provide a basis for qualifying software architecture-oriented CASE technology against a reference model of software architecture concepts; this would be analogous to the way CARDS qualifies command center components against a GCCA.

### **3.13.4 Anticipated System Demonstrations**

#### **3.13.4.1 WingZ**

WingZ is a commercial spreadsheet package and has very extensive graphic capabilities with 3-D imaging available as well as a Hyperscript language to enhance user documents. WingZ provides spreadsheet, drawing, hypermedia, and chart-making capabilities.

With the use of Datalink, WingZ can interface with Sybase and send SQL commands to the SQL server during a briefing session. Wingz is available for all the major graphical interfaces, making it very portable and usable.

#### **3.13.4.2 UNAS\_SALE**

SALE is a COTS CASE tool assisting the software architect in developing distributed applications utilizing the UNAS architectural design paradigm. Through SALE's graphical user interface, UNAS objects can be assembled using UNAS rules for building distributed applications. UNAS is a process-based, message-driven language framework for rapidly developing distributed applications. The final product of a SALE session is UNAS-specific Ada code for the management of distributed applications.

UNAS\_SALE, developed by TRW Systems Engineering and Development Division in Carson, CA, provides a full working version demonstration of the product. Due to licensing agreements, the only means to access the full working version SALE demonstration is by logging into Solitaire, the CARDS Library server, by users with a CARDS Unix account. UNAS\_SALE is not currently qualified into the Command Center Domain, although it is anticipated to undergo the CARDS qualification process in the future. At a later date, there will be a "screen capture" walk-through for AFS users. Currently a tutorial exists for the UNAS\_SALE component.

---

## APPENDIX A - References

[CARDS94a] CARDS Library User's Guide, STARS-AC-0006/000/00, Sept 93.

[CARDS94b] CARDS Version Description Document, STARS-AC-0B007/001/00, Sept 93.

[CARDS94c] Library Operation Policies and Procedures, Volume III - Library Development Handbook, Update - STARS-AC-04109/002/00, 28 Feb 94.

[CARDS94d] Technical Concepts Document, STARS-AC-03536/003/00, 28 Feb 94.

[CARDS93a] RLF User's Manual, Version 4.1, STARS-UC-05156/013/00, Mar 93.

[CARDS93b] RLF Modeler's Manual, RLF Version 4.1, STARS-UC-05156/011/00, Feb 93.

[CARDS93c] RLF Modeler Tutorial, STARS-UC-05156/020/00, Feb 93.

[CARDS92a] Command Center Domain Model Description, STARS-AC-04110/001/00, Nov 92.

[CARDS92b] RLF Graphical Browser User's Guide, STARS-SC-03065/004A/00, Jan 92.

[DCA90] Command Center System Architecture and TA/CE Guidance, Defense Communications Agency, Sept 90.

[DOD91] Multicommmand Required Operational Capability for Command Centers, MROC 1-89, DoD and Joint Staff, Feb 91.

[DISA91] Command Center Design Handbook, DISA, 91.

[DOD92] DoD Technical Reference Model, Version 1.3, Sept 92.

[ESD92] Generic Command Center Phase 2 Prototype Summary Report, ESD/AVS Hanscom AFB, Jan 92.

[GIAR92] CLIPS User's Guide, NASA, Joseph Giarratano, Sept 92.

[KANG90] Feature-Oriented Domain Analysis (FODA) Feasibility Study, SEI, Kang, Cohen, Hess, Novak, Peterson, Nov 90.

[NIST] Applications Portability Profile (APP) The U.S. Government's Open System Environment Profile, NIST APP Special Publication 500-187.

[PRIE92] Domain Analysis and Software Systems Modeling, IEEE Computer Society Press, Prieto-Diaz, Ruben and Arango, Guillermo, 92.

---

## APPENDIX B - Glossary of Terms

action	A mechanism permitting a user of the RLF graphical browser to invoke calls to the underlying operating system, including invoking other tools.
advice	An option offered to a user by the Reuse Library Framework's graphical browser which provides expert guidance by invoking appropriate inferencers to aid the user in navigating a model.
aggregation	Relationship between AdaKNET concepts which express attributes, characteristics, features (as the term is used in FODA [KANG90], functional capabilities, requirements or metrics; that is, any relationship that exists between two concepts which is not a specialization relationship.
application	A system which provides a set of general services for solving some type of user problem.
application platform	Hardware and software that provides services to higher level application software (e.g. operating system).
architecture modeling	The process of creating the software architecture(s) that implement(s) a solution to the problems in the domain.
automated message handling	Processing of strictly formatted messages.
category	see concept.
class	see concept.
command center	A facility from which a commander and her/his representatives direct operations and control forces. It is organized to gather, process, analyze, display and disseminate planning and operational data and to perform other related tasks.
component	A set of reusable resources that are related by virtue of being the inputs to various stages of the software design lifecycle, including requirements, design, code, test cases, documentation, etc. Components are the fundamental elements in a reusable software library.
component-based library	A library that is organized around a collection of reusable components. The underlying operational concept is that of search and retrieval of individual components.

	Components found in such libraries are classified in broad, generalized categories.
concept	An atomic unit of the AdaKNET knowledge representation scheme, representing an idea or thing, also known as a generic concept, a category or a class.
converged	Said of an AdaKNET role range for which the minimum and maximum have been set to the same value.
display	An aid to briefing such as video monitor or projection system.
domain	An area of activity or knowledge containing applications which share a set of common capabilities and data.
domain analysis	The process of identifying, collecting, organizing, analyzing and representing the relevant information in a domain, based on the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory and emerging technology within the domain.
domain criteria	Specifications that a potential component must adhere to in order to obtain acceptability in the domain and inclusion in the library. Domain criteria are a composite of three sets of constraints: component constraints, architectural constraints, and implementation constraints.
domain engineering	An encompassing process which includes domain analysis and the subsequent construction of components, methods, tools and supporting documentation that address the problems of system/subsystem development through the application of the knowledge in the domain model and software architecture.
domain model	A definition of the functions, objects, data and relationships in a domain, consisting of a concise representation of the commonalities and differences of the problems of the domain and their solutions (application programs).
domain modeling	The process of encoding knowledge about a domain into a formalism
entity	A particular and discrete unit; a named product, process, object or relationship.

external interfaces	The physical connection and protocol between the command center and other information systems.
feature	A prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems.
filler	A role used only between individuals. The filler of an individual's relationship must adhere to the relationship's restrictions. Both the owner and the "filler" must be individuals.
generic architecture	A collection of high-level paradigms and constraints that characterize the commonality and variances of the interactions and relationships between the various components in a system.
harvest	The retrieval of the source and/or executable code for components from the library into a specified directory during system composition.
individual	A knowledge representation of the reusable component of the library. An AdaKNET term for Represents a specific instantiation of a concept. Also known as an individual concept, an object or an instance.
individuation	The relationship between an individual to its subsuming concept. Indicates that an individual is an actual instance of the idea represented by the concept.
inferencer	(Also called rule-base) A mechanism which uses existing facts and rules about the elements of a model to perform reasoning about that model and deduce new facts and rules.
inherit	see inheritance.
inheritance	A mechanism whereby classes make use of the procedures, attributes, and /or data defined in other classes.
instance	see individual.
intermediate levels of specialization	Concepts that partition a category into subcategories.
knowledge-based software engineering	Software engineering with tools that utilize a knowledge representation of a domain and/or process.
library model	A model that represents the domain components and the relationships between them.

---

library modeling	The process of building a library model that accurately reflects the functionality and structure of the target domain.
meta-model	"model of modeling" or the knowledge-representation formalism used to encode library models.
model-based library	A library that is organized around the principle that what matters in a repository is the context in which reusable software components are used and the relationships among components. The focus of a model-based library is the model (requirements, architectures, design decisions and rationales) and the software which implements these models.
multilevel security	Information processing and communications which allow two or more classification levels of information to be processed simultaneously within the same system when some users are not cleared for all levels of information present.
object	see individual.
(role) name	Refers to the name of an AdaKNET aggregation relationship.
operator	A user who employs the resources of command center software to meet mission objectives.
(role) range	The number of simultaneous copies that may exist of an AdaKNET aggregation relationship.
reusable component	A component (including requirements, designs, code, test data, specifications, documentation, expertise etc.) designed and implemented for the specific purpose of being reused.
role	An AdaKNET term which refers to an aggregation ("consists of") relationship between two concepts.
role restriction	Refers to an inherited AdaKNET aggregation relationship which is narrowed at the inheriting concept, either by further restricting the range or by further restricting the type.
rule-base	(Also called inferencer) A collection of rules about the elements of a domain. A rule describes the relationships, requirements and constraints among components.

---

software architecture	High-level paradigms and constraints characterizing the structure of operations and objects, their interfaces and control to support the implementation of applications in a domain. Includes a description of each software component's functionality, name, parameters and their types, and a description of the components' interrelationships.
software reuse	The process of implementing new software systems using existing software information.
specialization	The act of declaring that one concept represents a narrowing of the idea represented by another concept.
standard descriptors	Basic conceptual units that form the interface between domain architectures and reusable components (i.e., high-level mini-specs for a class of components).
subsume	Having an is-a relation with a concept where the subsuming concept is a larger and more abstract category (e.g., X is_a Z and Y is_a Z therefore Z subsumes X and Y).
system composition	The automatic configuration of a prototype system based on hardware and software requirements.
taxonomy	The theory, principles and process of categorizing entities in established categories.
(role) type	The allowable range of values of an AdaKNET aggregation role.
workstation	An electronic hardware component of the command center through which the user can communicate, process information and prepare briefings.

## APPENDIX C - LIBRARY ACTIONS

Actions can be any executable command or script. If an "action" is included at a concept, the phrase **Perform Action** appears on the pop-up menu when the user chooses the node which represents that concept. Another pop-up menu appears when **Perform Action** is chosen. This menu displays the action or actions available at that concept. If there are no actions at the concept, the **Perform Action** option is not presented as a menu choice.

Many actions have been added to the model since the last version of this document. It is the intention that some actions, currently invoked at a limited number of concepts, will eventually be available at many others and that additional actions will be implemented.

The list of invocable actions and a short description follows:

- **Display Abstract** - Display an Abstract outlining the features and capabilities of an asset.
- **Display Relationships Graphically**-This function displays relationships of the library model in a graphical format.
- **Display Relationships Textually**-This function displays relationships of the library model in a textual format.
- **Extract Contents** - This function provides the ability to extract the contents of directories and/or files from an RLF Library to a user specified target directory.
- **Obtain Help** -This function provides the ability to view an ascii formatted file containing pertinent help information for a particular node.
- **Produce System** - This function calls the System Composition Tool.
- **Provide Description** - Displays the Description file for a particular node. The preview utility is used to display the Description file. It is available at all local concepts in the model.
- **Run Demo** -This function allows the user to execute a demonstration in a subshell. Demonstrations called through this action generally have a graphical user interface.
- **Show Assessment** - This function provides the user with the ability to view a Product Evaluation Report.
- **View\_message\_format** - This function displays information about message formats.
- **View Roadmap** - This function allows the user to view text describing major sections of the model.

- **Qualify Component-** This function provides the ability to view or obtain a printout of the optional and required features of the current category.
- **Picture Image-** This function provides the ability to view graphic images related to the current node.